Rewriting and narrowing for constructor systems with call-time choice semantics∗

FRANCISCO J. LÓPEZ-FRAGUAS, ENRIQUE MARTIN-MARTIN,
JUAN RODRÍGUEZ-HORTALÁ and JAIME SÁNCHEZ-HERNÁNDEZ
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
(*e-mail:* fraguas@sip.ucm.es, emartinm@fdi.ucm.es,
juan.rodriguez.hortala@gmail.com, jaime@sip.ucm.es)

submitted 30 November 2010; revised 14 November 2011; accepted 4 July 2012

Abstract

Non-confluent and non-terminating constructor-based term rewriting systems are useful for the purpose of specification and programming. In particular, existing functional logic languages use such kind of rewrite systems to define possibly non-strict non-deterministic functions. The semantics adopted for non-determinism is *call-time choice*, whose combination with non-strictness is a non trivial issue, addressed years ago from a semantic point of view with the Constructor-based Rewriting Logic (CRWL), a well-known semantic framework commonly accepted as suitable semantic basis of modern functional logic languages. A drawback of CRWL is that it does not come with a proper notion of one-step reduction, which would be very useful to understand and reason about how computations proceed. In this paper we develop thoroughly the theory for the first order version of let-rewriting, a simple reduction notion close to that of classical term rewriting, but extended with a let-binding construction to adequately express the combination of call-time choice with non-strict semantics. Let-rewriting can be seen as a particular textual presentation of term graph rewriting. We investigate the properties of let-rewriting, most remarkably their equivalence with respect to a conservative extension of the CRWL-semantics coping with let-bindings, and we show by some case studies that having two interchangeable formal views (reduction/semantics) of the same language is a powerful reasoning tool. After that, we provide a notion of let-narrowing which is adequate for call-time choice as proved by soundness and completeness results of let-narrowing with respect to let-rewriting. Moreover, we relate those let-rewriting and let-narrowing relations (and hence CRWL) with ordinary term rewriting and narrowing, providing in particular soundness and completeness of let-rewriting with respect to term rewriting for a class of programs which are deterministic in a semantic sense.
To appear in *Theory and Practice of Logic Programming* (TPLP).

KEYWORDS: term rewriting systems, constructor-based rewriting logic, narrowing, non-determinism, call-time choice semantics, sharing, local bindings

∗ This work has been partially supported by the Spanish projects FAST-STAMP (TIN2008-06622-C03-01/TIN), PROMETIDOS-CM (S2009TIC-1465) and GPD-UCM (UCM-BSCH-GR58/ 08-910502).

$$\begin{array}{ll} coin \to 0 & repeat(X) \ \to \ X : repeat(X) \\ coin \to 1 & heads(X : Y : Ys) \ \to \ (X, Y) \end{array}$$

Fig. 1. A non-terminating and non-confluent program

1 Introduction

Term rewriting systems (TRS, (Baader and Nipkow 1998)) are a well-known and useful formalism from the point of view of specification and programming. The theory of TRS underlies many of the proposals made in the last decades for so-called *functional logic programming*, attempting to integrate into a single language the main features of both functional and logic programming —see (DeGroot and Lindstrom 1986; Hanus 1994; Hanus 2007) for surveys corresponding to different historical stages of the development of functional logic languages—. Typically, functional logic programs are modeled by some kind of TRS to define functions, and logic programming capabilities are achieved by using some kind of *narrowing* as operational mechanism. Narrowing, a notion coming from the field of automated theorem proving, generalizes rewriting by using unification instead of matching in reduction steps. Up to 14 different variants of narrowing were identified in (Hanus 1994) as being used in different proposals for the integration of functional and logic programming.

Modern functional logic languages like *Curry* (Hanus et al. 1995; Hanus (ed.) 2006) or *Toy* (López-Fraguas and Sánchez-Hernández 1999; Caballero and Sánchez 2006) consider that programs are constructor-based term rewrite systems, possibly non-terminating and non-confluent, thus defining possibly non-strict non-deterministic functions. For instance, in the program of Figure 1, non-confluence comes from the two rules of *coin* and non-termination is due to the rule for *repeat*.

For non-determinism, those systems adopt *call-time choice* semantics (Hussmann 1993; González-Moreno et al. 1999), also called sometimes *singular* semantics (Søndergaard and Sestoft 1992). Loosely speaking, call-time choice means to pick a value for each argument of a function application before applying it. Call-time choice is easier to understand and implement in combination with strict semantics and eager evaluation in terminating systems as in (Hussmann 1993), but can be made also compatible —via partial values and sharing— with non-strictness and laziness in the presence of non-termination.

In the example of Figure 1 the expression $heads(repeat(coin))$ can take, under call-time choice, the values $(0, 0)$ and $(1, 1)$, but not $(0, 1)$ or $(1, 0)$. The example illustrates also a key point here: ordinary term rewriting (called *run-time choice* in (Hussmann 1993)) is an unsound procedure for call-time choice semantics, since a possible term rewriting derivation is:

$$\begin{array}{l} heads(repeat(coin)) \to heads(coin : repeat(coin)) \to \\ heads(0 : repeat(coin)) \to heads(0 : coin : repeat(coin)) \to \\ heads(0 : 1 : repeat(coin)) \to (0, 1) \end{array}$$

In operational terms, call-time choice requires to *share* the value of all copies of a given subexpression created during reduction (all the occurrences of *coin*, in the reduction above). In contrast, with ordinary term rewriting all copies evolve independently.

It is commonly accepted (see e.g. (Hanus 2007)) that call-time choice semantics combined with non-strict semantics is adequately formally expressed by the CRWL framework[1] (González-Moreno et al. 1996; González-Moreno et al. 1999), whose main component is a proof calculus that determines the semantics of programs and expressions. The flexibility and usefulness of CRWL is evidenced by the large set of extensions that have been devised for it, to cope with relevant aspects of declarative programming: higher order functions, types, constraints, constructive failure, ...; see (Rodríguez-Artalejo 2001) for a survey on the CRWL approach. However, a drawback of the CRWL-framework is its lack of a proper one-step reduction mechanism that could play a role similar to term rewriting with respect to equational logic. Certainly CRWL includes operational procedures in the form of goal-solving calculi (González-Moreno et al. 1999; Vado-Vírseda 2003) to solve so-called *joinability* conditions, but they are too complex to be seen as a basic way to explain or understand how a reduction can proceed in the presence of non-strict non-deterministic functions with call-time choice semantics.

On the other hand, other works have been more influential on the operational side of the field, specially those based on the notion of *needed narrowing* (Antoy et al. 1994; Antoy et al. 2000), a variant of narrowing that organizes the evaluation of arguments in function calls in an adequate way (optimal, for some classes of programs). Needed narrowing became the 'official' operational procedure of functional logic languages, and has also been subject of several variations and improvements (see (Hanus 2007; Escobar et al. 2005)).

These two coexisting branches of research (one based on CRWL, and the other based on classical term rewriting, mostly via needed narrowing) have remained disconnected for many years from the technical point of view, despite the fact that they both refer to what intuitively is the same programming language paradigm.

A major problem to establish the connection was that the theory underlying needed narrowing is classical term rewriting, which, as we saw above, is not valid for non-determinism with call-time choice semantics. This was not a flaw in the conception of needed narrowing, as it emerged in a time when non-deterministic functions had not yet started to play a distinctive role in the functional logic programming paradigm. The problem is overcome in practice by adding a sharing mechanism to the implementation of narrowing, using for instance standard Prolog programming techniques (Cheong and Fribourg 1993; Loogen et al. 1993; Antoy and Hanus 2000). But this is merely an implementation patch that cannot be used as a precise and sound technical basis for the application of results and techniques from the semantic side to the operational side and vice versa. Other works, specially (Echahed and Janodet 1998; Albert et al. 2005) have addressed in a more formal way the issue of sharing in functional logic programming, but they are not good

[1] CRWL stands for Constructor Based ReWriting Logic.

starting points to establish a relationship with the CRWL world (see 'Related work' below).

In (López-Fraguas et al. 2007b) we aimed at establishing a bridge, by looking for a new variant of term rewriting tailored to call-time choice as realized by CRWL, trying to fulfil the following requirements:

- it should be based on a notion of rewrite step useful to follow how a computation proceeds step by step.
- it should be simple enough to be easily understandable for non-expert potential users. (e.g., students or novice programmers) of functional logic languages adopting call-time choice.
- it should be provably equivalent to CRWL, as a well-established technical formulation of call-time choice.
- it should serve as a basis of subsequent notion of narrowing and evaluation strategies.

That was realized in (López-Fraguas et al. 2007b) by means of let-rewriting, a simple modification of term rewriting using local bindings in the form of let-expressions to express sharing. Let-rewriting will be fully presented in Section 4, but its main intuitions can be summarized as follows:

(i) do not rewrite a function call if any of its arguments is evaluable (i.e., still contains other function calls), even if there is a matching rule;
(ii) instead, extract those evaluable arguments to outer let-bindings of the form $let\ X = e\ in\ e'$;
(iii) if after some reduction steps the *definiens* e of the let-binding becomes a constructor term t —a value— then the binding X/t can be made effective in the body e'. In this way, the values obtained for e in the reduction are shared, and therefore call-time choice is respected.

Consider, for instance, the program example of Figure 1 and the expression

$$heads(repeat(coin))$$

for which we previously performed an ordinary term rewriting reduction ending in $(0, 1)$. Now we are going to apply liberally the previous intuitive hints as a first illustration of let-rewriting. Note first that no rewrite step using a program rule can be done with the whole expression *heads(repeat(coin))*, since in this case there is no matching rule. But we can extract the argument *repeat(coin)* to a let-binding, obtaining:

$$let\ X = repeat(coin)\ in\ heads(X)$$

Now we cannot rewrite *repeat(coin)*, even though the program rule for *repeat* matches it, because *coin* is evaluable. Again, we can create a let-binding for *coin*, that will be used to share the value selected for *coin*, if at any later step in the reduction *coin* is indeed reduced:

$$let\ Y = coin\ in\ let\ X = repeat(Y)\ in\ heads(X)$$

At this point there is no problem with rewriting *repeat(Y)*, which gives:

$$let\ Y = coin\ in\ let\ X = Y : repeat(Y)\ in\ heads(X)$$

Rewriting *repeat(Y)* again, we have:

$$let\ Y = coin\ in\ let\ X = Y : Y : repeat(Y)\ in\ heads(X)$$

Reducing *repeat(Y)* indefinitely leads to non-termination, but, at the same time, its presence inhibits the application of the binding for X. What we can do is creating a new let-binding for the remaining *repeat(Y)*, which results in:

$$let\ Y = coin\ in\ let\ Z = repeat(Y)\ in\ let\ X = Y : Y : Z\ in\ heads(X)$$

Now, the binding for X can be performed, obtaining:

$$let\ Y = coin\ in\ let\ Z = repeat(Y)\ in\ heads(Y : Y : Z)$$

At this point, we can use the rule for *heads* to evaluate $heads(Y : Y : Z)$, because nothing evaluable remains in its argument $Y : Y : Z$, arriving at:

$$let\ Y = coin\ in\ let\ Z = repeat(Y)\ in\ (Y, Y)$$

We proceed now by reducing *coin*, for instance, to 0 (reducing it to 1 is also possible):

$$let\ Y = 0\ in\ let\ Z = repeat(Y)\ in\ (Y, Y)$$

Performing the binding for Y leads to:

$$let\ Z = repeat(0)\ in\ (0, 0)$$

Since Z does not occur in $(0, 0)$, its binding is junk that could be deleted (there will be a rule for that in the definition of let-rewriting), and the reduction is finished yielding the value

$$(0, 0)$$

It is apparent that $(1, 1)$ is another possible result, but not $(0, 1)$ nor $(1, 0)$, a behavior coherent with call-time choice.

In this example we have tried to proceed in a more or less natural 'lazy' way. However, the previous intuitive precepts —and its complete and precise realization in Section 4— do not assume any particular strategy for organizing reductions, but only determine which are the 'legal movements' in call-time choice respectful reductions. Strategies have been left aside in the paper, not only for simplicity, but also to keep them independent of the basic rules for term rewriting with sharing (see however Section 6.2).

Let-rewriting was later on extended to cope with narrowing (López-Fraguas et al. 2009c) and higher order features (López-Fraguas et al. 2008).

This paper is a substantially revised and completed presentation of the theory of first order let-rewriting and let-narrowing proposed in (López-Fraguas et al. 2007b; López-Fraguas et al. 2009c); some contents have been also taken from (López-Fraguas et al. 2008). Here, we unify technically those papers and develop a deeper investigation of the properties of let-rewriting and related semantics issues.

Related work Our let-rewriting and let-narrowing relations are not the only nor the first formal operational procedures tuned up to accomplish with the call-time choice semantics of functional logic languages. We have already mentioned the goal-solving calculi associated to the CRWL-framework and its variants (González-Moreno et al. 1999; González-Moreno et al. 1997; Vado-Vírseda 2003).

A natural option to express different levels of sharing in rewriting is given by the theory of term graph rewriting (Barendregt et al. 1987; Plump 2001). In (Echahed and Janodet 1997; Echahed and Janodet 1998), the theory of needed rewriting and narrowing was extended to the framework of so-called admissible graph rewriting systems, aiming at formally modeling the operational behavior of functional logic programs. Originally, those works considered orthogonal systems, and extra variables were not allowed. These restrictions were dropped in (Antoy et al. 2007) (however, a formal treatment of the extension is missing).

As a matter of fact, our let-rewriting relation can be understood as a particular textual adaptation and presentation of term graph rewriting in which a shared node is made explicit in the syntax by giving it a name in a let-binding. The achievements of Echahed's works are somehow incomparable to ours, even if both are attempts to formalize sharing in constructor based systems. They focus and succeed on adapting known optimal strategies to the graph rewriting and narrowing setting; they also take profit of the fine-grained descriptions permitted by graphs to manage aspects of data structures like cycles or pointers. However, they do not try to establish a technical relationship with other formulations of call-time choice. In contrast, proving equivalence of our operational formalisms wrt. the CRWL semantic framework has been a main motivation of our work, but we do not deal with the issue of strategies, except for a short informal discussion at the end of the paper.

It is our thought that proving equivalence with respect to CRWL of term graph rewriting as given in (Echahed and Janodet 1997) would have been a task much harder than the route we follow here. We see a reason for it. The basic pieces that term rewriting and CRWL work with are purely syntactic: terms, substitutions, etc. Graph rewriting recast these notions in terms of graphs, homomorphisms, etc. In contrast, let-rewriting and let-narrowing keep the same set of basic pieces of term rewriting and CRWL. In this way, the formalisms are relatively close and moving from one to another becomes technically more natural and comfortable. This applies also to some further developments of our setting that we have made so far, like the extension to higher order features given in (López-Fraguas et al. 2008), the combination of semantics proposed in (López-Fraguas et al. 2009a), or the application of let-rewriting as underlying formal notion of reduction for type systems in functional logic languages (López-Fraguas et al. 2010b; López-Fraguas et al. 2010a).

Another proposal that can be seen as reformulation of graph rewriting was given in (Albert et al. 2005), inspired in Launchbury's natural semantics (Launchbury 1993) for lazy evaluation in functional programming. It presents two operational (natural and small-step) semantics for functional logic programs supporting sharing and residuation (a specific feature of Curry). These semantics use a flat representation of programs coming from an implicit program transformation encoding the demand analysis used by needed narrowing, and some kind of heaps to express bindings for variables. As in our case, let-expressions are used to express sharing. The approach is useful as a technical basis for implementation and program manipulation purposes; but we think that the approach is too low-level and close to a particular operational strategy to be a completely satisfactory choice as basic abstract reduction mechanism for call-time choice. In (López-Fraguas et al. 2007a) we established a technical relation of CRWL with the operational procedures of (Albert et al. 2005). But this turned out to be a really hard task, even if it was done only for a restricted class of programs and expressions.

Our work focuses on term rewriting systems as basic formalism, as happens with the majority of papers about the foundations of functional logic programming, in particular the CRWL-series. The idea of reformulating graph rewriting in a syntactic style by expressing sharing through let-bindings has been applied also to other contexts, most remarkably to λ-calculus considered as a basis of functional programming (Ariola and Arvind 1995; Ariola et al. 1995; Ariola and Felleisen 1997; Maraist et al. 1998). In a different direction, but still in relation with λ-calculus, other papers (Kutzner and Schmidt-Schauß 1998; Schmidt-Schauß and Machkasova 2008) have extended it with some kind of non-deterministic choice, an idea that comes back to McCarthy's *amb* (McCarthy 1963). As a final note, we should mention that our initial ideas about let-rewriting were somehow inspired by (López-Fraguas and Sánchez-Hernández 2001; Sánchez-Hernández 2004) where indexed unions of set expressions —a construction generalizing the idea of let-expressions —were used to express sharing in an extension of CRWL to deal with constructive failure.

The rest of the paper is organized as follows. Section 2 presents some preliminaries about term rewriting and the CRWL framework; although with them the paper becomes almost self-contained, some familiarity with the basic notions of TRS certainly help to read the paper. Section 3 contains a first discussion about failed or partial solutions to the problem of expressing non-strict call-time choice by a simple notion of rewriting. Section 4 is the central part of the paper. First, it introduces local bindings in the syntax to express sharing, defines let-rewriting as an adequate notion of rewriting for them and proves some intrinsic properties of let-rewriting. After that, in Section 4.2, we extend the CRWL-logic to a new CRWL$_{let}$-logic able to deal with lets in programs and expressions, and we investigate in depth the properties of the induced semantics, mostly through the notion of *hypersemantics*. Finally, in Section 4.3 we prove results of soundness and completeness of let-rewriting with respect to CRWL$_{let}$, which have as corollary the equivalence of both, and hence the equivalence of let-rewriting and CRWL for programs and expressions not containing lets, as the original CRWL ones are. Section 5 aims at showing the power of having reduction and semantics as equivalent interchangeable tools for reasoning, including a remarkable case study. In Section 6 we generalize the notion of let-rewriting to that of let-narrowing and give soundness and completeness results of the latter with respect to the former. At the end of the section we give some hints on how computations can be organized according to known narrowing strategies. Section 7 addresses the relationship between let-rewriting and classical term rewriting, proving in particular their equivalence for semantically deterministic programs. Finally, Section 8 analyzes our contribution and suggests further work. For the sake of readability, most of the (fully detailed) proofs have been moved to Appendix A.

2 Preliminaries

2.1 Constructor based term rewriting systems

We assume a fixed first order signature $\Sigma = CS \cup FS$, where CS and FS are two disjoint sets of constructor and defined function symbols respectively, each of them with an associated arity. We write CS^n and FS^n for the set of constructor and function symbols of arity n respectively, and Σ^n for $CS^n \cup FS^n$. As usual notations we write $c, d, \ldots$ for constructors, $f, g, \ldots$ for functions and $X, Y, \ldots$ for variables taken from a denumerable set $\mathcal{V}$. The notation $\overline{o}$ stands for tuples of any kind of syntactic objects.

The set *Exp* of *expressions* is defined as $Exp \ni e ::= X \mid h(e_1, \ldots, e_n)$, where $X \in \mathcal{V}$, $h \in \Sigma^n$ and $e_1, \ldots, e_n \in Exp$. The set $CTerm$ of *constructed terms* (or *c-terms*) has the same definition of *Exp*, but with h restricted to CS^n (so $CTerm \subsetneq Exp$). The intended meaning is that *Exp* stands for evaluable expressions, i.e., expressions that can contain (user-defined) function symbols, while $CTerm$ stands for data terms representing values. We will write $e, e', \ldots$ for expressions and $t, s, p, t', \ldots$ for c-terms. The set of variables occurring in an expression e will be denoted as $var(e)$.

Contexts (with one hole) are defined by $Cntxt \ni \mathcal{C} ::= [\ ] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$, where $h \in \Sigma^n$. The application of a context $\mathcal{C}$ to an expression e, written as $\mathcal{C}[e]$, is defined inductively as follows:

$$\begin{array}{rcl} [\ ][e] & = & e \\ h(e_1, \ldots, \mathcal{C}, \ldots, e_n)[e] & = & h(e_1, \ldots, \mathcal{C}[e], \ldots, e_n) \end{array}$$

Substitutions are finite mappings $\sigma : \mathcal{V} \longrightarrow Exp$ which extend naturally to $\sigma : Exp \longrightarrow Exp$. We write $e\sigma$ for the application of the substitution σ to e. The domain and variable range of a substitution σ are defined as $dom(\sigma) = \{X \in \mathcal{V} \mid X\sigma \neq X\}$ and $vran(\sigma) = \bigcup_{X \in dom(\sigma)} var(X\sigma)$. By $[X_1/e_1, \ldots, X_n/e_n]$ we denote the substitution σ such that $Y\sigma = e_i$ if $Y \equiv X_i$ for some $X_i \in \{X_1, \ldots, X_n\}$, and $Y\sigma = Y$ otherwise. Given a set of variables D, the notation $\sigma|_D$ represents the substitution σ restricted to D and $\sigma|_{\setminus D}$ is a shortcut for $\sigma|_{(\mathcal{V} \setminus D)}$. A *c-substitution* is a substitution θ such that $X\theta \in CTerm$ for all $X \in dom(\theta)$. We write *Subst* and *CSubst* for the sets of substitutions and c-substitutions respectively.

A *term rewriting system* is any set of rewrite rules of the form $l \to r$ where $l, r \in Exp$ and $l \notin \mathcal{V}$. A *constructor based rewrite rule* or *program rule* has the form $f(p_1, \ldots, p_n) \to r$ where $f \in FS^n$, $r \in Exp$ and $(p_1, \ldots, p_n)$ is a linear tuple of c-terms, where linear means that no variable occurs twice in the tuple. Notice that we allow r to have extra variables (i.e., variables not occurring in the left-hand side). To be precise, we say that X is an extra variable in the rule $l \to r$ iff $X \in var(r) \setminus var(l)$, and by $vExtra(R)$ we denote the set of extra variables in a rule R. Then a *constructor system* or *program* $\mathcal{P}$ is any set of program rules, i.e., a term rewriting system composed only of program rules.

Given a program $\mathcal{P}$, its associated *rewrite relation* $\rightarrow_{\mathcal{P}}$ is defined as $\mathcal{C}[l\sigma] \rightarrow_{\mathcal{P}} \mathcal{C}[r\sigma]$ for any context $\mathcal{C}$, rule $l \rightarrow r \in \mathcal{P}$ and $\sigma \in Subst$. There, the subexpression $l\sigma$ is called the redex used in that *rewriting step*. Notice that σ can instantiate extra variables to any expression. For any binary relation $\mathcal{R}$ we write $\mathcal{R}^*$ for the reflexive and transitive closure of $\mathcal{R}$, and $\mathcal{R}^n$ for the composition of $\mathcal{R}$ with itself n times. We write $e_1 \overset{*}{\rightarrow}_{\mathcal{P}} e_2$ for a term rewriting *derivation* or *reduction* from e_1 to e_2, and $e_1 \overset{n}{\rightarrow}_{\mathcal{P}} e_2$ for a n-step reduction. e_2 is a *normal form* wrt. $\rightarrow_{\mathcal{P}}$, written as $\downarrow^{\mathcal{P}} e_2$, if there is not any e_3 such that $e_2 \rightarrow_{\mathcal{P}} e_3$; and e_2 is a normal form for e_1 wrt. $\rightarrow_{\mathcal{P}}$, written as $e_1 \downarrow^{\mathcal{P}} e_2$, iff $e_1 \overset{*}{\rightarrow}_{\mathcal{P}} e_2$ and e_2 is a normal form. When presenting derivations, we will sometimes underline the redex used at each rewriting step. In the following, we will usually omit the reference to $\mathcal{P}$ when writing $e_1 \rightarrow_{\mathcal{P}} e_2$, or denote it by $\mathcal{P} \vdash e_1 \rightarrow e_2$.

A program $\mathcal{P}$ is confluent if for any $e, e_1, e_2 \in Exp$ such that $e \rightarrow^*_{\mathcal{P}} e_1$, $e \rightarrow^*_{\mathcal{P}} e_2$ there exists $e_3 \in Exp$ such that both $e_1 \rightarrow^*_{\mathcal{P}} e_3$ and $e_2 \rightarrow^*_{\mathcal{P}} e_3$.

2.2 The CRWL framework

We present here a simplified version of the CRWL framework (González-Moreno et al. 1996; González-Moreno et al. 1999). The original CRWL logic considered also the possible presence of *joinability* constraints as conditions in rules in order to give a better treatment of strict equality as a built-in, a subject orthogonal to the aims of this work. Furthermore, it is possible to replace conditions by the use of an *if_then* function, as has been technically proved in (Sánchez-Hernández 2004) for CRWL and in (Antoy 2005) for term rewriting. Therefore, we consider only unconditional program rules.

In order to deal with non-strictness at the semantic level, we enlarge Σ with a new constant (i.e., a 0-ary constructor symbol) $\perp$ that stands for the undefined value. The sets $Exp_\perp$, $CTerm_\perp$, $Subst_\perp$, $CSubst_\perp$ of partial expressions, etc., are defined naturally. Notice that $\perp$ does not appear in programs. Partial expressions are ordered by the *approximation* ordering $\sqsubseteq$ defined as the least partial ordering satisfying $\perp \sqsubseteq e$ and $e \sqsubseteq e' \Rightarrow \mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$ for all $e, e' \in Exp_\perp, \mathcal{C} \in Cntxt$. This partial ordering can be extended to substitutions: given $\theta, \sigma \in Subst_\perp$ we say $\theta \sqsubseteq \sigma$ if $X\theta \sqsubseteq X\sigma$ for all $X \in \mathcal{V}$.

The semantics of a program $\mathcal{P}$ is determined in CRWL by means of a proof calculus able to derive reduction statements of the form $e \rightarrowtriangle t$, with $e \in Exp_\perp$ and $t \in CTerm_\perp$, meaning informally that t is (or approximates to) a possible value of e, obtained by evaluating e using $\mathcal{P}$ under call-time choice.

The CRWL-proof calculus is presented in Figure 2. Rule **(B)** allows any expression to be undefined or not evaluated (non-strict semantics). Rule **(OR)** expresses that to evaluate a function call we must choose a compatible program rule, perform parameter passing (by means of a c-substitution θ) and then reduce the right-hand side. The use of c-substitutions in (OR) is essential to express call-time choice; notice also that by the effect of θ in (OR), extra variables in the right-hand side of a

$$\textbf{(B)}\ \frac{}{e \rightarrowtriangle \perp} \qquad \textbf{(RR)}\ \frac{}{X \rightarrowtriangle X}\quad X \in \mathcal{V}$$

$$\textbf{(DC)}\ \frac{e_1 \rightarrowtriangle t_1\ \ldots\ e_n \rightarrowtriangle t_n}{c(e_1, \ldots, e_n) \rightarrowtriangle c(t_1, \ldots, t_n)} \quad c \in CS^n$$

$$\textbf{(OR)}\ \frac{e_1 \rightarrowtriangle p_1\theta \ldots\ e_n \rightarrowtriangle p_n\theta\ \ r\theta \rightarrowtriangle t}{f(e_1, \ldots, e_n) \rightarrowtriangle t} \quad \begin{array}{l}(f(p_1, \ldots, p_n) \rightarrow r) \in \mathcal{P}\\ \theta \in CSubst_\perp\end{array}$$

Fig. 2. Rules of CRWL

$$\dfrac{\dfrac{\dfrac{\dfrac{}{0 \rightarrowtriangle 0}DC}{\mathbf{coin} \rightarrowtriangle \mathbf{0}}OR \quad \dfrac{\dfrac{}{0 \rightarrowtriangle 0}DC \quad \dfrac{\dfrac{}{0 \rightarrowtriangle 0}DC \quad \dfrac{\dfrac{}{0 \rightarrowtriangle 0}DC \quad \dfrac{}{repeat(0) \rightarrowtriangle \perp}B}{0 : repeat(0) \rightarrowtriangle 0 : \perp}DC}{repeat(0) \rightarrowtriangle 0 : \perp}OR}{0 : repeat(0) \rightarrowtriangle 0 : 0 : \perp}DC}{repeat(coin) \rightarrowtriangle 0 : 0 : \perp}OR \quad \dfrac{\dfrac{}{0 \rightarrowtriangle 0}DC \quad \dfrac{}{0 \rightarrowtriangle 0}DC}{(0,0) \rightarrowtriangle (0,0)}DC}{heads(repeat(coin)) \rightarrowtriangle (0,0)}OR$$

Fig. 3. A CRWL-derivation for $heads(repeat(coin)) \rightarrowtriangle (0, 0)$

rule can be replaced by any partial c-term, but not by any expression as in ordinary term rewriting $\rightarrow_{\mathcal{P}}$. We write $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$ to express that $e \rightarrowtriangle t$ is derivable in the CRWL-calculus using the program $\mathcal{P}$, but in many occasions we will omit the mention to $\mathcal{P}$, writing simply $e \rightarrowtriangle t$.

Definition 1 (*CRWL-denotation*)
Given a program $\mathcal{P}$, the *CRWL-denotation* of an expression $e \in Exp_\perp$ is defined as

$$[\![e]\!]^{\mathcal{P}}_{CRWL} = \{t \in CTerm_\perp \mid \mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t\}$$

We will usually omit the subscript CRWL and/or the superscript $\mathcal{P}$ when implied by the context.

As an example, Figure 3 shows a *CRWL-derivation* or *CRWL-proof* for the statement $heads(repeat(coin)) \rightarrowtriangle (0, 0)$, using the program of Figure 1. Observe that in the derivation there is only one reduction statement for *coin* (namely $coin \rightarrowtriangle 0$), and the obtained value 0 is then *shared* in the whole derivation, as corresponds to call-time choice.

In alternative derivations, *coin* could be reduced to 1 (or to $\perp$). It is easy to check that:

$$[\![heads(repeat(coin))]\!] = \{(0, 0), (1, 1), (\perp, 0), (0, \perp), (\perp, 1), (1, \perp), (\perp, \perp), \perp\}$$

Note that $(1, 0), (0, 1) \notin [\![heads(repeat(coin))]\!]$.

We stress the fact that the CRWL-calculus *is not* an operational mechanism for executing programs, but a way of describing the logic of programs. In particular, the rule (B) is a semantic artifact to reflect in a CRWL-proof of a statement $e \rightarrowtriangle t$ the fact that, for obtaining t as value of e, one does not need to know the value of a certain subexpression e' (to which the rule (B) is applied). But the calculus comes with no indication of when to apply (B) in a successful proof. At the operational level, the CRWL framework is accompanied with various lazy narrowing-based goal-solving calculi (González-Moreno et al. 1999; Vado-Vírseda 2003) not considered in this paper.

One of the most important properties of CRWL is its compositionality, a property very close to the DET-additivity property for algebraic specifications of (Hussmann 1993) or the referential transparency property of (Sondergaard and Sestoft 1990). Compositionality shows that the CRWL-denotation of any expression placed in a context only depends on the CRWL-denotation of that expression. This implies that the semantics of a whole expression depends only on the semantics of its constituents, as shown by the next result, which is an adaptation of a similar one proved for the higher order case in (López-Fraguas et al. 2008).

Theorem 1 (*Compositionality of CRWL*)
For any $\mathcal{C} \in Cntxt$, $e, e' \in Exp_\perp$

$$[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!]$$

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt. [\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$

According to this result we can express for example

$$[\![heads(repeat(coin))]\!] = \bigcup_{t \in [\![coin]\!]} \bigcup_{s \in [\![repeat(t)]\!]} [\![heads(s)]\!]$$

The right hand side has an intuitive reading that reflects call-time choice: get a value t of *coin*, then get a value s of $repeat(t)$ and then get a value of $heads(s)$.

In Theorem 2 we give an alternative formulation to the compositionality property. Although it is essentially equivalent to Theorem 1, it is a somehow more abstract statement, based on the notion of *denotation of a context* introduced in Definition 2. Our main reason for developing such alternative is to give good insights for the compositionality results of the extension of CRWL to be presented in Section 4.3.

We will use sometimes *Den* as an alias for $\mathcal{P}(CTerm_\perp)$, i.e, for the kind of objects that are CRWL-denotations of expressions[2]. We define the denotation of a context $\mathcal{C}$ as a denotation transformer that reflects call-time choice.

Definition 2 (*Denotation of a context*)
Given $\mathcal{C} \in Cntxt$, its denotation is a function $[\![\mathcal{C}]\!] : Den \to Den$ defined as

$$[\![\mathcal{C}]\!]\delta = \bigcup_{t \in \delta} [\![\mathcal{C}[t]]\!], \ \forall \delta \in Den$$

With this notion, compositionality can be trivially re-stated as follows:

[2] *Den* is indeed a superset of the set of actual denotations, which are particular elements of $\mathcal{P}(CTerm_\perp)$, namely *cones* —see (González-Moreno et al. 1999)—. But this is not relevant to the use we make of *Den*.

Theorem 2 (*Compositionality of CRWL, version 2*)
For any $\mathcal{C} \in Cntxt$ and $e, e' \in Exp_\perp$

$$[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}]\!][\![e]\!]$$

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$

The formulation of compositionality given by Theorem 2 makes even more apparent than Theorem 1 the fact that the syntactic decomposition of an expression e in the form $\mathcal{C}[e']$ has a direct semantic counterpart, in the sense that the semantics of e is determined by the semantics of its syntactic constituents $\mathcal{C}$ and e'. However, Theorems 1 and 2 are indeed of the same strength, since each of them can be easily proved from the other.

3 CRWL and rewriting: a first discussion

Before presenting let-rewriting we find interesting to discuss a couple of (in principle) shorter solutions to the problem of expressing non-strict call-time choice semantics by means of a simple one-step reduction relation. A first question is whether a new relation is needed at all: maybe call-time choice can be expressed by ordinary term rewriting via a suitable program transformation. The next result shows that in a certain technical sense this is not possible: due to different closedness under substitution and compositionality properties of call-time choice and term rewriting, none of them can be naturally simulated by each other.

Proposition 1
There is a program $\mathcal{P}$ for which the following two conditions hold:

i) no term rewriting system (constructor based or not) $\mathcal{P}'$ verifies

$$\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t \text{ iff } \mathcal{P}' \vdash e \to^* t \text{ , for all } e \in Exp, t \in CTerm$$

ii) no program $\mathcal{P}'$ verifies

$$\mathcal{P} \vdash e \to^* t \text{ iff } \mathcal{P}' \vdash_{CRWL} e \rightarrowtriangle t \text{ , for all } e \in Exp, t \in CTerm$$

Proof
The following simple program $\mathcal{P}$ suffices:

$$f(X) \to c(X, X) \qquad coin \to 0 \qquad coin \to 1$$

i) We reason by contradiction. Assume there is a term rewriting system $\mathcal{P}'$ such that: $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t \Leftrightarrow e \to^*_{\mathcal{P}'} t$, for all e, t. Since $\mathcal{P} \vdash_{CRWL} f(X) \rightarrowtriangle c(X, X)$, we must have $f(X) \to^*_{\mathcal{P}'} c(X, X)$. Now, since $\to^*_{\mathcal{P}'}$ is closed under substitutions (Baader and Nipkow 1998), we have $f(coin) \to^*_{\mathcal{P}'} c(coin, coin)$, and therefore$f(coin) \to^*_{\mathcal{P}'} c(coin, coin) \to^*_{\mathcal{P}'} c(0, 1)$. But it is easy to see that $\mathcal{P} \vdash_{CRWL} f(coin) \rightarrowtriangle c(0, 1)$ does not hold.

ii) Assume now there is a program $\mathcal{P}'$ such that: $\mathcal{P} \vdash e \to^* t \Leftrightarrow \mathcal{P}' \vdash_{CRWL} e \rightarrowtriangle t$, for all e, t. Since $\mathcal{P} \vdash f(coin) \to^* c(0, 1)$, we have $\mathcal{P}' \vdash_{CRWL} f(coin) \rightarrowtriangle c(0, 1)$. By compositionality of call-time choice (Theorem 1), there must exist a possibly partial

$(\mathbf{B}^{rw})$	$\mathcal{C}[e]$	$\rightarrowtail$	$\mathcal{C}[\bot]$	$\forall \mathcal{C} \in \mathit{Cntxt}, e \in \mathit{Exp}_\bot$
$(\mathbf{OR}^{rw})$	$\mathcal{C}[f(t_1\theta, \ldots, t_n\theta)]$	$\rightarrowtail$	$\mathcal{C}[r\theta]$	$\forall \mathcal{C} \in \mathit{Cntxt}, f(t_1, \ldots, t_n) \to r \in \mathcal{P}$, $\theta \in \mathit{CSubst}_\bot$

Fig. 4. A one-step reduction relation for non-strict call-time choice

$t \in CTerm_\bot$ such that $\mathcal{P}' \vdash_{CRWL} coin \rightarrowtriangle t$ and $\mathcal{P}' \vdash_{CRWL} f(t) \rightarrowtriangle c(0,1)$. Now we distinguish cases on the value of t:

(a) If $t \equiv \bot$, then monotonicity of $CRWL$-derivability —see (González-Moreno et al. 1999) or Proposition 3 below— proves that $\mathcal{P}' \vdash_{CRWL} f(s) \rightarrowtriangle c(0,1)$ for any $s \in CTerm_\bot$, in particular $\mathcal{P}' \vdash_{CRWL} f(0) \rightarrowtriangle c(0,1)$. Then, by the assumption on $\mathcal{P}'$, it should be $\mathcal{P} \vdash f(0) \to^* c(0,1)$, but this is not true.

(b) If $t \equiv 0$, then $\mathcal{P}' \vdash f(0) \rightarrowtriangle c(0,1)$ as before. The cases $t \equiv 1$, $t \equiv Y$ or $t \equiv d(\overline{s})$ for a constructor d different from $0, 1$ lead to similar contradictions.

□

Notice that Proposition 1 does not make any assumption about signatures: in any of *i) or ii)*, no extension of the signature can lead to a simulating $\mathcal{P}'$. This does not contradict Turing completeness of term rewriting systems. Turing completeness arguments typically rely on encodings not preserving the structure of data, something not contemplated in Proposition 1.

In a second trial, requiring minimal changes over ordinary term rewriting, we impose that the substitution θ in a rewriting step must be a c-substitution, as in the rule (OR) of CRWL. This is done in the one-step rule $(\mathbf{OR}^{rw})$ in Figure 4. According to it, the step $heads(repeat(coin)) \to heads(coin : repeat(coin))$ in the introductory example of Figure 1 would not be legal anymore. However, (OR^{rw}) corresponds essentially to innermost evaluation, and is not enough to deal with non-strictness, as the following example shows:

Example 1

Consider the rules $f(X) \to 0$ and $loop \to loop$. With a non-strict semantics $f(loop)$ should be reducible to 0. But (OR^{rw}) does not allow the step $f(loop) \to 0$; only $f(loop) \to f(loop) \to \ldots$ is a valid (OR^{rw})-reduction, thus leaving $f(loop)$ semantically undefined, as would correspond to a strict semantics.

At this point, the rule (B) of CRWL is a help, since it allows to discard the evaluation of any (sub)-expression by reducing it to $\bot$. The result of this discussion is the one-step reduction relation $\rightarrowtail$ given in Figure 4.

This relation satisfies our initial goals to a partial extent, as it is not difficult to prove the following equivalence result.

Theorem 3

Let $\mathcal{P}$ be a CRWL-program, $e \in Exp_\bot$ and $t \in CTerm_\bot$. Then:

$$\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t \ \mathit{iff}\ e \rightarrowtail^*_{\mathcal{P}} t$$

This result has an interesting reading: non-strict call-time choice can be achieved via innermost evaluation if at any step one has the possibility of reducing a subexpression to $\perp$ (then, we could speak also of *call-by-partial value*). For instance, a $\rightarrowtail$-rewrite sequence with the example of Figure 1 would be:

$$\begin{array}{l} heads(repeat(coin)) \rightarrowtail heads(repeat(0)) \rightarrowtail \\ heads(0 : repeat(0)) \rightarrowtail heads(0 : 0 : repeat(0)) \rightarrowtail \\ heads(0 : 0 :\perp) \rightarrowtail (0, 0) \end{array}$$

This gives useful intuitions about non-strict call-time choice and can actually serve for a very easy implementation of it, but has a major drawback: in general, reduction of a subexpression e requires a *don't know* guessing between (B^{rw}) and (OR^{rw}), because at the moment of reducing e it is not known whether its value will be needed or not later on in the computation. Instead of reducing to $\perp$, let-rewriting will create a let-binding *let* U=e *in* $\ldots$, which does not imply any guessing and keeps e for its eventual future use.

4 Rewriting with local bindings

Inspired by (Ariola and Arvind 1995; Ariola et al. 1995; Ariola and Felleisen 1997; Maraist et al. 1998; Plump 1998; Sánchez-Hernández 2004), let-rewriting extends the syntax of expressions by adding local bindings to express sharing and call-time choice. Formally the syntax for *let-expressions* is:

$$LExp \ni e ::= X \mid h(e_1, \ldots, e_n) \mid let\ X = e_1\ in\ e_2$$

where $X \in \mathcal{V}$, $h \in \Sigma^n$, and $e_1, e_2, \ldots, e_n \in LExp$. The intended behaviour of $let\ X = e_1\ in\ e_2$ is that the expression e_1 will be reduced only once (at most) and then its corresponding value will be shared within e_2. For $let\ X = e_1\ in\ e_2$ we call e_1 the *definiens* of X, and e_2 the *body* of the let-expression.

The sets $FV(e)$ of *free* and $BV(e)$ *bound* variables of $e \in LExp$ are defined as:

$$\begin{array}{l} FV(X) = \{X\} \\ FV(h(\overline{e})) = \bigcup_{e_i \in \overline{e}} FV(e_i) \\ FV(let\ X = e_1\ in\ e_2) = FV(e_1) \cup (FV(e_2) \backslash \{X\}) \end{array}$$

$$\begin{array}{l} BV(X) = \emptyset \\ BV(h(\overline{e})) = \bigcup_{e_i \in \overline{e}} BV(e_i) \\ BV(let\ X = e_1\ in\ e_2) = BV(e_1) \cup BV(e_2) \cup \{X\} \end{array}$$

Notice that with the given definition of $FV(let\ X = e_1\ in\ e_2)$ there are not recursive let-bindings in the language since the possible occurrences of X in e_1 are not considered bound and therefore refer to a 'different' X. For example, the expression $let\ X = f(X)\ in\ g(X)$ can be equivalently written as $let\ Y = f(X)\ in\ g(Y)$. This is similar to what is done in (Maraist et al. 1998; Ariola et al. 1995; Ariola and Felleisen 1997), but not in (Albert et al. 2005; Launchbury 1993). Recursive lets have their own interest but there is not a general consensus in the functional logic community about their meaning in presence of non-determinism. We remark also that the let-bindings introduced by let-rewriting derivations to be presented in Section 4.1 are not recursive. Therefore, recursive lets are not considered in this work.

We will use the notation *let* $\overline{X = a}$ *in* e as a shortcut for *let* $X_1 = a_1$ *in* $\ldots$ *in let* $X_n = a_n$ *in* e. The notion of *one-hole context* is also extended to the new syntax:

$$\mathcal{C} ::= [\,] \mid let\ X = \mathcal{C}\ in\ e \mid let\ X = e\ in\ \mathcal{C} \mid h(\ldots, \mathcal{C}, \ldots)$$

By default, we will use contexts with lets from now on.

Free variables of contexts are defined as for expressions, so that $FV(\mathcal{C}) = FV(\mathcal{C}[h])$, for any $h \in \Sigma^0$. However, the set $BV(\mathcal{C})$ of *variables bound by a context* is defined quite differently because it consists only of those let-bound variables visible from the hole of $\mathcal{C}$. Formally:

$$\begin{array}{l} BV([\,]) = \emptyset \\ BV(h(\ldots, \mathcal{C}, \ldots)) = BV(\mathcal{C}) \\ BV(let\ X = e\ in\ \mathcal{C}) = \{X\} \cup BV(\mathcal{C}) \\ BV(let\ X = \mathcal{C}\ in\ e) = BV(\mathcal{C}) \end{array}$$

As a noticeable difference with respect to (López-Fraguas et al. 2007b), from now on we will allow to use lets in any program, so our program rules have the shape $f(p_1, \ldots, p_n) \to r$, for $f \in FS^n$, $(p_1, \ldots, p_n)$ a linear tuple of c-terms, and $r \in LExp$. Notice, however, that the notion of c-term does not change: c-terms do not contain function symbols nor lets, although they can contain bound variables when put in an appropriate context as happens for example with the subexpression (X, X) in the expression *let* $X = coin$ *in* (X, X).

As usual with syntactical binding constructs, we assume a variable convention according to which bound variables can be consistently renamed as to ensure that the same variable symbol does not occur free and bound within an expression. Moreover, to keep simple the management of substitutions, we assume that whenever θ is applied to an expression $e \in LExp$, the necessary renamings are done in e to ensure $BV(e) \cap (dom(\theta) \cup vran(\theta)) = \emptyset$. With all these conditions the rules defining application of substitutions are simple while avoiding variable capture:

$$\begin{array}{l} X\theta = \theta(X),\ \text{for } X \in \mathcal{V} \\ h(e_1, \ldots, e_n)\theta = h(e_1\theta, \ldots, e_n\theta),\ \text{for } h \in \Sigma^n \\ (let\ X = e_1\ in\ e_2)\theta = let\ X = e_1\theta\ in\ e_2\theta \end{array}$$

The following example illustrates the use of these conventions.

$$\begin{array}{l} (let\ X = c(X)\ in\ let\ Y = z\ in\ d(X, Y))[X/c(Y)] \\ = (let\ U = c(X)\ in\ let\ V = z\ in\ d(U, V))[X/c(Y)] \\ = let\ U = c(c(Y))\ in\ let\ V = z\ in\ d(U, V) \end{array}$$

The following substitution lemma will be often a useful technical tool:

Lemma 1 (*Substitution lemma for let-expressions*)
Let $e, e' \in LExp_\perp$, $\theta \in Subst_\perp$ and $X \in \mathcal{V}$ such that $X \notin dom(\theta) \cup vran(\theta)$. Then:

$$(e[X/e'])\theta \equiv e\theta[X/e'\theta]$$

(Fapp) $f(p_1, \ldots, p_n)\theta \to^l r\theta$, if $(f(p_1, \ldots, p_n) \to r) \in \mathcal{P}, \theta \in CSubst$

(LetIn) $h(\ldots, e, \ldots) \to^l let\ X = e\ in\ h(\ldots, X, \ldots)$,
if $h \in \Sigma$, $e \equiv f(\overline{e'})$ with $f \in FS$ or $e \equiv let\ Y = e'\ in\ e''$, and X is a fresh variable

(Bind) $let\ X = t\ in\ e \to^l e[X/t]$, if $t \in CTerm$

(Elim) $let\ X = e_1\ in\ e_2 \to^l e_2$, if $X \notin FV(e_2)$

(Flat) $let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \to^l let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3)$
if $Y \notin FV(e_3)$

(Contx) $\mathcal{C}[e] \to^l \mathcal{C}[e']$,
if $\mathcal{C} \neq [\ ]$, $e \to^l e'$ using any of the previous rules, and in case $e \to^l e'$ is a (Fapp) step using $(f(\overline{p}) \to r) \in \mathcal{P}$ and $\theta \in CSubst$, then $vran(\theta|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$.

Fig. 5. Rules of the let-rewriting relation $\to^l$

4.1 The let-rewriting relation

Let-expressions can be reduced step by step by means of the *let-rewriting* relation $\to^l$, shown in Figure 5. Rule (**Contx**) allows us to use any subexpression as redex in the derivation. (**Fapp**) performs a rewriting step in the proper sense, using a program rule. Note that only c-substitutions are allowed, to avoid copying of unevaluated expressions which would destroy sharing and call-time choice. To prevent that the restriction of (**Fapp**) to total c-substitutions results in a strict semantics, we also provide the rule (**LetIn**) that suspends the evaluation of a subexpression by introducing a let-binding. If its value is needed later on, its evaluation can be performed by some (**Contx**) steps and the result propagated by (**Bind**). This latter rule is safe wrt. call-time choice because it only propagates c-terms, that is, either completely defined values (without any bound variable) or partially computed values with some suspension (bound variable) on it, which will be safely managed by the calculus. On the other hand, if the bound variable disappears from the body of the let-binding during evaluation, rule (**Elim**) can be used for garbage collection. This rule is useful to ensure that normal forms corresponding to values are c-terms. Finally, (**Flat**) is needed for flattening nested lets; otherwise some reductions could become wrongly blocked or forced to diverge. Consider for example the program $\{loop \to loop, g(s(X)) \to 1\}$ and the expression $g(s(loop))$, which can be reduced to $let\ X = (let\ Y = loop\ in\ s(Y))\ in\ g(X)$ by applying (**LetIn**) twice. Then, without (**Flat**) we could only perform reduction steps on *loop*, thus diverging; by using (**Flat**), we can obtain $let\ Y = loop\ in\ let\ X = s(Y)\ in\ g(X)$, which can be finally reduced to 1 by applying (**Bind**), (**Fapp**) and (**Elim**). The condition $Y \notin FV(e_3)$ in (**Flat**) could be dropped by the variable convention, but we have included it to keep the rules independent of the convention. Quite different is the case of (**Elim**), where the condition $X \notin FV(e_2)$ is indeed necessary.

Note that, in contrast to CRWL or the relation $\rightarrowtail$ in Section 3, let-rewriting does not need to use the semantic value $\bot$, which does not appear in programs nor in computations.

Example 2
Consider the program of Figure 1. We can perform the following let-rewriting deriva-

tion for the expression $heads(repeat(coin))$, where in each step the corresponding redex has been underlined for the sake of readability.

$$\begin{array}{ll}
\underline{heads(repeat(coin))} & (LetIn)\\
\to^{l}\ let\ X = \underline{repeat(coin)}\ in\ heads(X) & (LetIn)\\
\to^{l}\ \underline{let\ X = (let\ Y = coin\ in\ repeat(Y))\ in\ heads(X)} & (Flat)\\
\to^{l}\ let\ Y = coin\ in\ let\ X = \underline{repeat(Y)}\ in\ heads(X) & (Fapp)\\
\to^{l}\ let\ Y = coin\ in\ let\ X = \underline{Y : repeat(Y)}\ in\ heads(X) & (LetIn)\\
\to^{l}\ let\ Y = coin\ in\ \underline{let\ X = (let\ Z = repeat(Y)\ in\ Y : Z)\ in\ heads(X)} & (Flat)\\
\to^{l}\ let\ Y = coin\ in\ let\ Z = repeat(Y)\ in\ \underline{let\ X = Y : Z\ in\ heads(X)} & (Bind)\\
\to^{l}\ let\ Y = coin\ in\ let\ Z = \underline{repeat(Y)}\ in\ heads(Y : Z) & (Fapp)\\
\to^{l}\ let\ Y = coin\ in\ let\ Z = \underline{Y : repeat(Y)}\ in\ heads(Y : Z) & (LetIn)\\
\to^{l}\ let\ Y = coin\ in\ \underline{let\ Z = (let\ U = repeat(Y)\ in\ Y : U)\ in\ heads(Y : Z)} & (Flat)\\
\to^{l}\ let\ Y = coin\ in\ let\ U = repeat(Y)\ in\ \underline{let\ Z = Y : U\ in\ heads(Y : Z)} & (Bind)\\
\to^{l}\ let\ Y = coin\ in\ let\ U = repeat(Y)\ in\ \underline{heads(Y : Y : U)} & (Fapp)\\
\to^{l}\ let\ Y = coin\ in\ \underline{let\ U = repeat(Y)\ in\ (Y, Y)} & (Elim)\\
\to^{l}\ let\ Y = \underline{coin}\ in\ (Y, Y) & (Fapp)\\
\to^{l}\ \underline{let\ Y = 0\ in\ (Y, Y)} & (Bind)\\
\to^{l}\ (0, 0) &
\end{array}$$

Note that there is not a unique $\to^{l}$-reduction leading to $(0, 0)$. The definition of $\to^{l}$, like traditional term rewriting, does not prescribe any particular strategy. The definition of on-demand evaluation strategies for let-rewriting is out of the scope of this paper, and is only informally discussed in Section 6.2.

We study now some properties of let-rewriting that have intrinsic interest and will be useful when establishing a relation to CRWL in next sections.

The same example used in Proposition 1 to show that CRWL is not closed under general substitutions shows also that the same applies to let-rewriting. However, let-rewriting is closed under c-substitutions, as expected in a semantics for call-time choice.

Lemma 2 (*Closedness under $CSubst$ of let-rewriting*)
For any $e, e' \in LExp$, $\theta \in CSubst$ we have that $e \to^{l\,n} e'$ implies $e\theta \to^{l\,n} e'\theta$.

Another interesting matter is the question of what happens in let-rewriting derivations in which the rule (Fapp) is not used—and as a consequence, the program is ignored.

Definition 3 (*The $\to^{lnf}$ relation*)
The relation $\to^{lnf}$ is defined by the rules of Figure 5 except (Fapp). As a consequence, for any program $\to^{lnf} \subseteq \to^{l}$.

We can think about any let-expression e as an expression from Exp in which some additional sharing information has been encoded using the let-construction. When we avoid the use of the rule (Fapp) in derivations, we do not make progress in the evaluation of the implicit let-less expression corresponding to e, but we change the sharing-enriched representation of that expression in the let-rewriting syntax. Following terminology from term graph rewriting —as in fact a let-expression is a textual representation of a term graph— all the rules of let-rewriting except (Fapp) move between two isomorphic term graphs (Plump 2001; Plump 1998). The $\rightarrow^{lnf}$ relation will be used to reason about these kind of derivations.

The first interesting property of $\rightarrow^{lnf}$ is that it is a terminating relation.

Proposition 2 (*Termination of* $\rightarrow^{lnf}$)
For any program $\mathcal{P}$, the relation $\rightarrow^{lnf}$ is terminating. As a consequence, every $e \in LExp$ has at least one $\rightarrow^{lnf}$-normal form e' (written as $e \downarrow^{lnf} e'$).

However, for nontrivial signatures the relation $\rightarrow^{lnf}$ is not confluent (hence the relation $\rightarrow^{l}$ is not confluent either).

Example 3
Consider a signature such that $f, g \in FS^0, c \in CS^2$ and $f \not\equiv g$. Then $c(f,g) \rightarrow^{lnf^*}$ $let\ X = f\ in\ let\ Y = g\ in\ c(X,Y)$ and $c(f,g) \rightarrow^{lnf^*}\ let\ Y = g\ in\ let\ X = f\ in\ c(X,Y)$, but these expressions do not have a common reduct.

The lack of confluence of let-rewriting is alleviated by a strong semantic property of $\rightarrow^{lnf}$ which, combined with the adequacy to CRWL of let-rewriting that we will see below, may be used as a substitute for confluence in some situations. These questions will be treated in detail in Section 4.3.1.

The next result characterizes $\rightarrow^{lnf}$-normal forms. What we do in $\rightarrow^{lnf}$ derivations is exposing the computed part of e —its outer constructor part— concentrating it in the body of the resulting let, that is, the part which is not a function application whose evaluation is pending. This is why we call it *'Peeling lemma'*.

Lemma 3 (*Peeling lemma*)
For any $e, e' \in LExp$, if $e \downarrow^{lnf} e'$ then e' has the shape $e' \equiv let\ \overline{X = f(\overline{t})}\ in\ e''$ such that $e'' \in \mathcal{V}$ or $e'' \equiv h(\overline{t'})$ with $h \in \Sigma$, $\overline{f} \subseteq FS$ and $\overline{t}, \overline{t'} \subseteq CTerm$.
Moreover if $e \equiv h(e_1, \ldots, e_n)$ with $h \in \Sigma$, then

$$e \equiv h(e_1, \ldots, e_n) \rightarrow^{lnf^*} let\ \overline{X = f(\overline{t})}\ in\ h(t_1, \ldots, t_n) \equiv e'$$

under the conditions above, and verifying also that $t_i \equiv e_i$ whenever $e_i \in CTerm$.

The next property of $\rightarrow^{l}$ and $\rightarrow^{lnf}$ uses the notion of *shell* $|e|$ of an expression e, that is the partial c-term corresponding to the outer constructor part of e. More precisely:

Definition 4 (*Shell of a let-expression*)

$$\begin{array}{lcll} |X| & = & X & \text{for } X \in \mathcal{V} \\ |c(e_1, \ldots, e_n)| & = & c(|e_1|, \ldots, |e_n|) & \text{for } c \in CS \\ |f(e_1, \ldots, e_n)| & = & \perp & \text{for } f \in FS \\ |let\ X = e_1\ in\ e_2| & = & |e_2|[X/|e_1|] & \end{array}$$

Notice that in the case of a let-rooted expression, the information contained in the binding is taken into account for building up the shell of the whole expression: for instance $|c(let\ X = 2\ in\ s(X))| = c(s(2))$.

During a computation, the evolution of shells reflects the progress towards a value. The next result shows that shells never decrease. Moreover, only (Fapp) may change shells. As discussed above, 'peeling' steps (i.e. $\rightarrow^{lnf}$- steps) just modify the representation of the implicit term graph corresponding to a let-expression; thus, they preserve the shell.

Lemma 4 (*Growing of shells*)

i) $e \rightarrow^{l^*} e'$ implies $|e| \sqsubseteq |e'|$, for any $e, e' \in LExp$
ii) $e \rightarrow^{lnf^*} e'$ implies $|e| \equiv |e'|$, for any $e, e' \in LExp$

4.2 ***The*** *$CRWL_{let}$* ***logic***

In this section we extend the CRWL logic to deal with let-expressions, obtaining an enlarged framework that will be useful as a bridge to establish the connection between CRWL and let-rewriting.

As in the CRWL framework, we consider partial let-expressions $e \in LExp_\perp$, defined in the natural way. The approximation order $\sqsubseteq$ is also extended to $LExp_\perp$ but now using the notion of context for let-expressions, which in particular implies that $let\ X = e_1\ in\ e_2 \sqsubseteq let\ X = e_1'\ in\ e_2'$ iff $e_1 \sqsubseteq e_1'$ and $e_2 \sqsubseteq e_2'$. The CRWL$_{let}$ logic results of adding the following rule (**Let**) to the CRWL logic of Section 2.2:

$$(\mathbf{Let})\ \frac{e_1 \rightarrow\!\!\triangleright\, t_1 \quad e_2[X/t_1] \rightarrow\!\!\triangleright\, t}{let\ X = e_1\ in\ e_2 \rightarrow\!\!\triangleright\, t}$$

We write $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrow\!\!\triangleright\, t$ if $e \rightarrow\!\!\triangleright\, t$ is derivable in the CRWL$_{let}$-calculus using the program $\mathcal{P}$. In many occasions, we will omit $\mathcal{P}$.

Definition 5 (CRWL$_{let}$*-denotation*)
Given a program $\mathcal{P}$, the *CRWL$_{let}$-denotation* of $e \in LExp_\perp$ is defined as:

$$[\![e]\!]^{\mathcal{P}}_{CRWL_{let}} = \{t \in CTerm_\perp \mid \mathcal{P} \vdash_{CRWL_{let}} e \rightarrow\!\!\triangleright\, t\}$$

We will omit the sub(super)-scripts when they are clear by the context.

There is an obvious relation between CRWL and CRWL$_{let}$ for programs and expressions without lets:

Theorem 4 (*CRWL vs. CRWL$_{let}$*)
For any program $\mathcal{P}$ without lets, and any $e \in Exp_\perp$:

$$[\![e]\!]^{\mathcal{P}}_{CRWL} = [\![e]\!]^{\mathcal{P}}_{CRWL_{let}}$$

This result allows us to skip the mention to CRWL or CRWL$_{let}$ when referring to the denotation $[\![e]\!]$ of an expression: if some let-binding occurs in e —or in the program wrt. which the denotation is considered— then $[\![e]\!]$ can be interpreted only as $[\![e]\!]_{CRWL_{let}}$; otherwise, both denotations coincide.

The CRWL$_{let}$ logic inherits from CRWL a number of useful properties.

Lemma 5
For any program $e \in LExp_\perp$, $t, t' \in CTerm_\perp$:

i) $t \rightarrow\!\!\triangleright\, t'$ iff $t' \sqsubseteq t$.
ii) $|e| \in [\![e]\!]$.

iii) $[\![e]\!] \subseteq (|e|\!\uparrow)\!\downarrow$, where for a given $E \subseteq LExp_\perp$ its upward closure is $E\!\uparrow = \{e' \in LExp_\perp | \; \exists e \in E.\; e \sqsubseteq e'\}$, its downward closure is $E\!\downarrow = \{e' \in LExp_\perp | \; \exists e \in E.\; e' \sqsubseteq e\}$, and those operators are overloaded for let-expressions as $e\!\uparrow = \{e\}\!\uparrow$ and $e\!\downarrow = \{e\}\!\downarrow$.

The first part of the previous result shows that c-terms can only be reduced to smaller c-terms. The other parts express that the shell of an expression represents 'stable' information contained in the expression in a similar way to Lemma 4, as the shell is in the denotation by *ii)*, and everything in the denotation comes from refining it by *iii)*.

The following results are adaptations to CRWL$_{let}$ of properties known for CRWL (González-Moreno et al. 1999; Vado-Vírseda 2002). The first one states that if we can compute a value for an expression then from greater expressions we can reach smaller values. The second one says that CRWL$_{let}$-derivability is closed for partial c-substitutions.

Proposition 3 (*Polarity of CRWL$_{let}$*)
For any $e, e' \in LExp_\perp$, $t, t' \in CTerm_\perp$, if $e \sqsubseteq e'$ and $t' \sqsubseteq t$ then $e \rightarrowtriangle t$ implies $e' \rightarrowtriangle t'$ with a proof of the same size or smaller—where the size of a CRWL$_{let}$-proof is measured as the number of rules of the calculus used in the proof.

Proposition 4 (*Closedness under c-substitutions of CRWL$_{let}$*)
For any $e \in LExp_\perp$, $t \in CTerm_\perp$, $\theta \in CSubst_\perp$, $t \in [\![e]\!]$ implies $t\theta \in [\![e\theta]\!]$.

Compositionality is a more delicate issue. Theorem 1 does not hold for CRWL$_{let}$, as shown by the following example: consider the program $\{f(0) \to 1\}$, the expression $e \equiv f(X)$ and the context $\mathcal{C} \equiv let\ X = 0\ in\ []$. $\mathcal{C}[e]$ can produce the value 1. However, $f(X)$ can only be reduced to $\perp$, and $\mathcal{C}[\perp]$ cannot reach the value 1. The point in that example is that the subexpression e needs some information from the context to produce a value that is then used by the context to compute the value for the whole expression $\mathcal{C}[e]$. This information may only be the definientia of some variables of e that get bound when put in $\mathcal{C}$; with this idea in mind we can state the following weak compositionality result for CRWL$_{let}$.

Theorem 5 (*Weak Compositionality of CRWL$_{let}$*)
For any $\mathcal{C} \in Cntxt$, $e \in LExp_\perp$

$$[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!] \qquad \text{if } BV(\mathcal{C}) \cap FV(e) = \emptyset$$

As a consequence, $[\![let\ X = e_1\ in\ e_2]\!] = \bigcup_{t_1 \in [\![e_1]\!]} [\![e_2[X/t_1]]\!]$.

In spite of not being a fully general compositionality result, Theorem 5 can be used to prove new properties of CRWL$_{let}$, like the following monotonicity property related to substitutions, that will be used later on. It is formulated for the partial order $\sqsubseteq$ over $LSubst_\perp$ (defined naturally as it happened for $Susbt_\perp$) and the preorder $\trianglelefteq$ over $LSubst_\perp$, defined by $\sigma \trianglelefteq \sigma'$ iff $\forall X \in \mathcal{V}, [\![\sigma(X)]\!] \subseteq [\![\sigma'(X)]\!]$.

Proposition 5 (*Monotonicity for substitutions of CRWL$_{let}$*)
If $\sigma \sqsubseteq \sigma'$ or $\sigma \trianglelefteq \sigma'$ then $[\![e\sigma]\!] \subseteq [\![e\sigma']\!]$, for any $e \in LExp_\perp$ and $\sigma, \sigma' \in LSubst_\perp$.

The limitations of Theorem 5 make us yearn for another semantic notion for let-expressions with a better compositional behaviour. We have already seen that the problem with CRWL$_{let}$ is the possible loss of definientia when extracting an expression from its context. But in fact what bound variables need is access to the *values* of their corresponding definientia, as it is done in the rule (Let) where the value of the definiens is transmited to the body of the let-binding by applying a c-substitution replacing the bound variable by that value. With these ideas in mind we define the stronger notion of *hyperdenotation* (sometimes we say *hypersemantics*), which gives a more active role to variables in expressions: in contrast to the denotation of an expression e, which is a set of c-terms, its hyperdenotation $[\![\![e]\!]\!]$ is a function mapping c-substitutions to denotations, i.e., to sets of c-terms.

Definition 6 (*Hyperdenotation*)
The hyperdenotation of an expression $e \in LExp_\perp$ under a program $\mathcal{P}$ is a function $[\![\![e]\!]\!]^{\mathcal{P}} : CSubst_\perp \to Den$ defined by $[\![\![e]\!]\!]^{\mathcal{P}}\ \theta = [\![e\theta]\!]^{\mathcal{P}}$.

As usual, in most cases we will omit the mention to $\mathcal{P}$. We will use sometimes HD as an alias for $CSubst_\perp \to Den$, i.e, for the kind of objects that are hyperdenotations of expressions.

The notion of hyperdenotation is strictly more powerful than the notion of CRWL$_{let}$ denotation. Equality of hyperdenotations implies equality of denotations —because if $[\![\![e]\!]\!] = [\![\![e']\!]\!]$ then $[\![e]\!] = [\![e\epsilon]\!] = [\![\![e]\!]\!]\epsilon = [\![\![e']\!]\!]\epsilon = [\![e'\epsilon]\!] = [\![e']\!]$— but the opposite does not hold: consider the program $\{f(0) \to 1\}$ and the expressions $f(X)$ and $\perp$; they have the same denotation (the set $\{\perp\}$) but different hyperdenotations, as $[\![\![\perp]\!]\!][X/0] \not\ni 1 \in [\![\![f(X)]\!]\!][X/0]$. Hypersemantics are useful to characterize the meaning of expressions present in a context in which some of its variables may get bound, like in the body of a let-binding or in the right hand side of a program rule. Therefore are useful to reason about expressions put in arbitrary contexts, in which let-bindings may freely appear.

Most remarkably, hyperdenotations allow to recover strong compositionality results for let-expressions similar to Theorems 1 and 2. We find it more intuitive to start the analog to the latter. Semantics of contexts were defined as denotation transformers (Definition 2). Analogously, the hypersemantics $[\![\![\mathcal{C}]\!]\!]$ of a context $\mathcal{C}$ is a hyperdenotation transformer defined as follows:

Definition 7 (*Hypersemantics of a context*)
Given $\mathcal{C} \in Cntxt$, its hyperdenotation is a function $[\![\![\mathcal{C}]\!]\!] : HD \to HD$ defined by induction over the structure of $\mathcal{C}$ as follows:

- $[\![\![[\,]]\!]\!]\varphi\theta = \varphi\theta$
- $[\![\![h(e_1, \ldots, \mathcal{C}, \ldots, e_n)]\!]\!]\varphi\theta = \bigcup\limits_{t \in [\![\![\mathcal{C}]\!]\!]\varphi\theta} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!]$
- $[\![\![let\ X = \mathcal{C}\ in\ e]\!]\!]\varphi\theta = \bigcup\limits_{t \in [\![\![\mathcal{C}]\!]\!]\varphi\theta} [\![let\ X = t\ in\ e\theta]\!]$
- $[\![\![let\ X = e\ in\ \mathcal{C}]\!]\!]\varphi\theta = \bigcup\limits_{t \in [\![\![e]\!]\!]\theta} [\![\![\mathcal{C}]\!]\!]\varphi(\theta[X/t])$

With this notion, our first version of strong compositionality for hypersemantics looks like Theorem 2.

Theorem 6 (*Compositionality of hypersemantics*)
For all $\mathcal{C} \in Cntxt$, $e \in LExp_\perp$

$$[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}]\!][\![e]\!]$$

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$.

This result implies that in any context we can replace any subexpression by another one having the same hypersemantics (and therefore also the same semantics) without changing the hypersemantics (hence the semantics) of the global expression.

In Theorems 2 and 6 the role of call-time choice is hidden in the definition of semantics and hypersemantics of a context, respectively. To obtain a version of strong compositionalty of hypersemantics closer to Theorem 1 and 5, we need some more notions and notations about hyperdenotations or, more generally, about functions in HD. Since they are set-valued functions, many usual set operations and relations can be lifted naturally in a pointwise manner to HD. The precise definitions become indeed clearer if we give them for general sets, abstracting away the details about HD. We introduce also some notions about decomposing set-valued functions that will be useful for hyperdenotations. We use freely λ-notation to write down a function in the mathematical sense; we may write $\lambda x \in A$ to indicate its domain A, if it not clear by the context.

Definition 8 (*Operations and relations for set-valued functions*)
Let A, B be two sets, $\mathcal{F}$ the set of functions $A \to \mathcal{P}(B)$, and $f, g \in \mathcal{F}$. Then:

i) The *hyperunion* of f, g is defined as $f \Cup g = \lambda x \in A.f(x) \cup g(x)$.

ii) More generally, the *hyperunion of a family* $\mathcal{I} \subseteq \mathcal{F}$, written indistinctly as $\Cup \mathcal{I}$ or $\Cup_{f \in \mathcal{I}} f$, is defined as

$$\Cup \mathcal{I} \equiv \Cup_{f \in \mathcal{I}} f =_{def} \lambda x \in A. \bigcup_{f \in \mathcal{I}} f(x)$$

Notice that $f \Cup g = \Cup \{f, g\}$.

iii) We say that f is *hyperincluded* in g, written $f \Subset g$, iff $\forall x \in A.f(x) \subseteq g(x)$.

iv) A *decomposition* of f is any $\mathcal{I} \subseteq \mathcal{F}$ such that $\Cup \mathcal{I} = f$.

v) The *elemental decomposition* of f is the following set of functions of $\mathcal{F}$:

$$\Delta f = \{\lambda x \in A. \begin{cases} \{b\} \text{ if } x = a \\ \emptyset \text{ otherwise} \end{cases} \mid a \in A, b \in f(a)\}$$

Or, using the abbreviation $\hat{\lambda} a.\{b\}$ as a shorthand for $\lambda x. \begin{cases} \{b\} \text{ if } x = a \\ \emptyset \text{ otherwise} \end{cases}$,

$$\Delta f = \{\hat{\lambda} a.\{b\} \mid a \in A, b \in f(a)\}$$

Decompositions are used to split set-valued functions into smaller pieces; elemental decompositions do it with minimal ones. For instance, if $f : \{a, b\} \to \mathcal{P}(\{0, 1, 2\})$ is given by $f(a) = \{0, 2\}$ and $f(b) = \{1, 2\}$, then $\Delta f = \{\hat{\lambda} a.\{0\}, \hat{\lambda} a.\{2\}, \hat{\lambda} b.\{1\}, \hat{\lambda} b.\{2\}\}$.

Hyperinclusion and hyperunion share many properties of standard set inclusion and union. Some of them are collected in the next result, that refer also to decompositions:

Proposition 6
Consider two sets A, B, and let $\mathcal{F}$ be the set of functions $A \to \mathcal{P}(B)$. Then:

i) $\Subset$ is indeed a partial order on $\mathcal{F}$, and Δf is indeed a decomposition of $f \in \mathcal{F}$, i.e., $\bigcupplus\ (\Delta f) = f$.

ii) Monotonicity of hyperunion wrt. inclusion: for any $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{F}$

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \text{ implies } \bigcupplus\ \mathcal{I}_1 \Subset \bigcupplus\ \mathcal{I}_2$$

iii) Distribution of unions: for any $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{F}$

$$\bigcupplus\ (\mathcal{I}_1 \cup \mathcal{I}_2) = (\bigcupplus\ \mathcal{I}_1) \Cup (\bigcupplus\ \mathcal{I}_2)$$

iv) Monotonicity of decomposition wrt. hyperinclusion: for any $f_1, f_2 \in \mathcal{F}$

$$f_1 \Subset f_2 \text{ implies } \Delta f_1 \subseteq \Delta f_2$$

We will apply all these notions, notations and properties to the case when $A \equiv CSubst_\perp$ and $B \equiv CTerm_\perp$ (i.e. $\mathcal{P}(B) \equiv Den$ and therefore $\mathcal{F} \equiv HD$). Therefore, we can speak of the hyperunion of two hyperdenotations, or of a family of them, we can elementarily decompose a hyperdenotation, etc.

Proposition 7 (*Distributivity under context of hypersemantics unions*)

$$[\![\mathcal{C}]\!](\bigcupplus\ H) = \bigcupplus_{\varphi \in H} [\![\mathcal{C}]\!]\varphi$$

With this result we can easily prove our desired new version of a strong compositionality result for hypersemantics, with a style closer to the formulations of Theorems 1 and 5. This new form of compositionality will be used in Section 5.1 for building a straightforward proof of the adequacy of a transformation that otherwise becomes highly involved by using other techniques.

Theorem 7 (*Compositionality of hypersemantics, version 2*)
For any $\mathcal{C} \in Cntxt$, $e \in LExp_\perp$:

$$[\![\mathcal{C}[e]]\!] = \bigcupplus_{\varphi \in H} [\![\mathcal{C}]\!]\varphi, \text{ for any decomposition } H \text{ of } [\![e]\!]$$

In particular: $[\![\mathcal{C}[e]]\!] = \bigcupplus_{\varphi \in \Delta [\![e]\!]} [\![\mathcal{C}]\!]\varphi$.

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt. [\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$.

Proof

$$\begin{array}{ll} [\![\mathcal{C}[e]]\!] = [\![\mathcal{C}]\!][\![e]\!] & \text{by compositionality } \textit{v.1} \text{ (Theorem 6)} \\ = [\![\mathcal{C}]\!](\Cup\ H) & \text{by definition of decomposition (Def. 8 } \textit{iv)} \\ = \Cup_{\varphi \in H} [\![\mathcal{C}]\!]\varphi & \text{by distributivity (Proposition 7)} \end{array}$$

□

As happened with Theorems 1 and 2 with respect to denotations, Theorems 6 and 7 are different aspects of the same property, which shows that the hypersemantics of a whole let-expression depends only on the hypersemantics of its constituents; it also allows us to interchange in a context any pair of expressions with the same hypersemantics. This is reflected on the fact that we have attached $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$ as a trivial consequence both in Theorem 6 and Theorem 7. Moreover, Theorem 6 can also be proved by a combination of Theorem 7 and Propositions 6 *i)* and 7, in a similar way to the proof for Theorem 7 above.

$$\begin{array}{ll} [\![\mathcal{C}[e]]\!] = \Cup_{\varphi \in H} [\![\mathcal{C}]\!]\varphi & \text{by compositionality } \textit{v.2} \text{ (Theorem 7)} \\ = [\![\mathcal{C}]\!](\Cup\ (\Delta[\![e]\!])) & \text{by distributivity (Proposition 7)} \\ = [\![\mathcal{C}]\!][\![e]\!] & \text{because } \Delta[\![e]\!] \text{ decomposes } e \text{ (Proposition 6 } \textit{i)}) \end{array}$$

Therefore Theorems 6 and 7 are results with the same strength, two sides of the same coin that will be useful tools for reasoning with hypersemantics.

To conclude, we present the following monotonicity property under contexts of hypersemantics, which will be useful in the next section.

Lemma 6 (*Monotonicity under contexts of hypersemantics*)
For any $\mathcal{C} \in Cntxt, \varphi_1, \varphi_2 \in HD$:

$$\varphi_1 \Subset \varphi_2 \text{ implies that } [\![\mathcal{C}]\!]\varphi_1 \Subset [\![\mathcal{C}]\!]\varphi_2$$

Proof
Assume $\varphi_1 \Subset \varphi_2$. Then:

$$\begin{array}{ll} [\![\mathcal{C}]\!]\varphi_1 = [\![\mathcal{C}]\!](\Cup\ (\Delta\varphi_1)) & \text{by Proposition 6 } \textit{i)} \\ = [\![\mathcal{C}]\!](\Cup\ \{\hat{\lambda}\mu.\{t\} \mid \mu \in CSubst_\perp, t \in \varphi_1\mu\}) & \text{by definition of } \Delta \\ \Subset [\![\mathcal{C}]\!](\Cup\ \{\hat{\lambda}\mu.\{t\} \mid \mu \in CSubst_\perp, t \in \varphi_2\mu\}) & \text{by Proposition 6 } \textit{ii)} \\ = [\![\mathcal{C}]\!](\Cup\ (\Delta\varphi_2)) & \text{by definition of } \Delta \\ = [\![\mathcal{C}]\!]\varphi_2 & \text{by Proposition 6 } \textit{i)} \end{array}$$

□

We have now the tools needed to tackle the task of formally relating CRWL and let-rewriting.

4.3 Equivalence of let-rewriting to CRWL and $CRWL_{let}$

In this section we prove soundness and completeness results of let-rewriting with respect to CRWL$_{let}$ and CRWL.

4.3.1 Soundness

Concerning soundness we want to prove that $\rightarrow^l$-steps do not create new CRWL-semantic values. More precisely:

Theorem 8 (*Soundness of let-rewriting*)
For all $e, e' \in LExp$, if $e \rightarrow^{l^*} e'$ then $[\![e']\!] \subseteq [\![e]\!]$.

Notice that because of non-determinism $\subseteq$ cannot be replaced by $=$ in this theorem. For example, with the program $\mathcal{P} = \{coin \rightarrow 0, coin \rightarrow 1\}$ we can perform the step $coin \rightarrow^l 0$, for which $[\![0]\!] = \{0, \perp\}$, $[\![coin]\!] = \{0, 1, \perp\}$.

It is interesting to explain why a direct reasoning with denotations fails to prove Theorem 8.

A proof could proceed straightforwardly by a case distinction on the rules for $\rightarrow^l$ to prove the soundness of a single $\rightarrow^l$ step. The problem is that the case for a (Contx) step would need the following monotonicity property under context of CRWL$_{let}$ denotations:

$$[\![e]\!] \subseteq [\![e']\!] \text{ implies } [\![\mathcal{C}[e]]\!] \subseteq [\![\mathcal{C}[e']]\!]$$

Unfortunately, the property is false, for the same reasons that already explained the weakness of Theorem 5: the possible capture of variables when switching from e to $\mathcal{C}[e]$.

Counterexample 1
Consider the program $\mathcal{P} = \{f(0) \rightarrow 1\}$. We have $[\![f(X)]\!] = \{\perp\} \subseteq \{\perp, 0\} = [\![0]\!]$, but when these expressions are placed within the context $let\ X = 0\ in\ [\ ]$ we obtain $[\![let\ X = 0\ in\ f(X)]\!] = \{\perp, 1\} \not\subseteq \{\perp, 0\} = [\![let\ X = 0\ in\ 0]\!]$.

The good thing is that we can overcome these problems by using hypersemantics. Theorem 8 will be indeed an easy corollary of the following generalization to hypersemantics.

Theorem 9 (*Hyper-Soundness of let-rewriting*)
For all $e, e' \in LExp$, if $e \rightarrow^{l^*} e'$ then $[\![[e']]\!] \Subset [\![[e]]\!]$.

And, in order to prove this generalized theorem, we also devise a generalization of the faulty monotonicity property of CRWL$_{let}$ denotations above mentioned. That generalization is an easy consequence of the compositionality and monotonicity under contexts of hypersemantics.

Lemma 7
For all $e, e' \in LExp_\perp$ and $\mathcal{C} \in Cntxt$, if $[\![e]\!] \Subset [\![e']\!]$ then $[\![[\mathcal{C}[e]]]\!] \Subset [\![[\mathcal{C}[e']]]\!]$.

Proof

$$\begin{array}{ll} [\![[\mathcal{C}[e]]]\!] = [\![[\mathcal{C}]]\!][\![e]\!] & \text{by Theorem 6} \\ \Subset [\![[\mathcal{C}]]\!][\![e']\!] & \text{by Lemma 6, as } [\![e]\!] \Subset [\![e']\!] \\ = [\![[\mathcal{C}[e']]]\!] & \text{by Theorem 6} \end{array}$$

□

With the help of Lemma 7, we can now prove Theorem 9 by a simple case distinction on the rules for $\rightarrow^{l}$ and a trivial induction on the length of the derivation. Now, Theorem 8 follows as an easy consequence.

Proof for Theorem 8
Assume $e \rightarrow^{l^*} e'$. By Theorem 9 we have $[\![e']\!] \Subset [\![e]\!]$, and therefore $[\![e'\theta]\!] \subseteq [\![e\theta]\!]$ for every $\theta \in CSubst_{\perp}$. Choosing $\theta = \epsilon$ (the empty substitution) we obtain $[\![e']\!] \subseteq [\![e]\!]$ as desired. □

The moral then is that *when reasoning about the semantics of expressions and programs with lets it is usually better to lift the problem to the hypersemantic world*, and then particularize to semantics the obtained result. This is done, for instance, in the following result:

Proposition 8 (*The $\rightarrow^{lnf}$ relation preserves hyperdenotation*)
For all $e, e' \in LExp$, if $e \ \rightarrow^{lnf^*} \ e'$ then $[\![e]\!] = [\![e']\!]$—and therefore $[\![e]\!] = [\![e']\!]$.

This result mirrors semantically the fact that $\rightarrow^{lnf}$ performs transitions between let-expressions corresponding to the same implicit term graph. Proposition 8 in some sense lessens the importance of the lack of confluence for the $\rightarrow^{lnf}$ relation seen in Section 4.1. Preservation of hyperdenotation may be used in some situations as a substitute for confluence, specially taking into account that let-rewriting and CRWL$_{let}$ enjoy a really strong equivalence, as it is shown in this section.

Finally, we combine the previous results in order to get our main result concerning the soundness of let-rewriting with respect to the CRWL$_{let}$ calculus:

Theorem 10 (*Soundness of let-rewriting*)
For any program $\mathcal{P}$ and $e \in LExp$ we have:

i) $e \rightarrow^{l\,*} e'$ implies $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle |e'|$, for any $e' \in LExp$.
ii) $e \rightarrow^{l\,*} t$ implies $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$, for any $t \in CTerm$.

Furthermore, if neither $\mathcal{P}$ nor e have lets then we also have:

iii) $e \rightarrow^{l\,*} e'$ implies $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle |e'|$, for any $e' \in LExp$.
iv) $e \rightarrow^{l\,*} t$ implies $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$, for any $t \in CTerm$.

Proof

i) Assume $e \rightarrow^{l\,*} e'$. Then, by Theorem 8 we have $[\![e']\!]_{CRWL_{let}} \subseteq [\![e]\!]_{CRWL_{let}}$. Since $|e'| \in [\![e']\!]_{CRWL_{let}}$ by Lemma 5, we get $|e'| \in [\![e]\!]_{CRWL_{let}}$, which means $e \rightarrowtriangle |e'|$.
ii) Trivial by *(i)*, since $|t| = t$ for any $t \in CTerm$.
iii) Just combining *i)* and Theorem 4.
iv) Just combining *ii)* and Theorem 4.

□

4.3.2 Completeness

Now we look for the reverse implication of Theorem 10, that is, the completeness of let-rewriting as its ability to compute, for any given expression, any value that can been computed by the CRWL-calculi. With the aid of the Peeling Lemma 3 we can prove the following strong completeness result for let-rewriting, which still has a certain technical nature.

Lemma 8 (*Completeness lemma for let-rewriting*)
For any $e \in LExp$ and $t \in CTerm_\perp$ such that $t \not\equiv \perp$,

$$e \rightarrowtriangle t \text{ implies } e \to^{l^*} let\ \overline{X = a}\ in\ t'$$

for some $t' \in CTerm$ and $\overline{a} \subseteq LExp$ such that $t \sqsubseteq |let\ \overline{X = a}\ in\ t'|$ and $|a_i| = \perp$ for every $a_i \in \overline{a}$. As a consequence, $t \sqsubseteq t'[\overline{X/\ \perp}]$.

Note the condition $t \not\equiv \perp$ is essential for this lemma to be true, as we can see by taking $\mathcal{P} = \{loop \to loop\}$ and $e \equiv loop$: while $loop \rightarrowtriangle \perp$, the only $LExp$ reachable from $loop$ is $loop$ itself.

Our main result concerning completeness of let-rewriting follows easily from Lemma 8. It shows that any c-term computed by CRWL or CRWL$_{let}$ for an expression can be refined by a let-rewriting derivation; moreover, if the c-term is total, then it can be exactly reached by let-rewriting.

Theorem 11 (*Completeness of let-rewriting*)
For any program $\mathcal{P}$, $e \in LExp$, and $t \in CTerm_\perp$ we have:

i) $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$ implies $e \to^{l\,*} e'$ for some $e' \in LExp$ such that $t \sqsubseteq |e'|$
ii) Besides, if $t \in CTerm$ then $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$ implies $e \to^{l\,*} t$

Furthermore, if neither $\mathcal{P}$ nor e have lets then we also have

iii) $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$ implies $e \to^{l\,*} e'$ for some $e' \in LExp$ such that $t \sqsubseteq |e'|$
iv) Besides, if $t \in CTerm$ then $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$ implies $e \to^{l\,*} t$

Proof
Regarding part *i)*, if $t \equiv \perp$ then we are done with $e \to^{l\,0} e$ as $\forall e, \perp \sqsubseteq |e|$. On the other hand, if $t \not\equiv \perp$ then by Lemma 8 we have $e \to^{l\,*} let\ \overline{X} = \overline{a}\ in\ t'$ such that $t \sqsubseteq |let\ \overline{X} = \overline{a}\ in\ t'|$.

To prove part *ii)*, assume $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$. Then, by Lemma 8, we get $e \to^{l\,*} let\ \overline{X} = \overline{a}\ in\ t'$ such that $t \sqsubseteq |let\ \overline{X} = \overline{a}\ in\ t| \equiv t'[\overline{X/\perp}]$, for some $t' \in CTerm, \overline{a} \subseteq LExp$. As $t \in CTerm$ then t is maximal wrt. $\sqsubseteq$, so $t \sqsubseteq t'[\overline{X/\perp}]$ implies $t'[\overline{X/\perp}] \equiv t$, but then $t'[\overline{X/\perp}] \in CTerm$ so it must happen that $FV(t') \cap \overline{X} = \emptyset$ and therefore $t' \equiv t'[\overline{X/\perp}] \equiv t$. But then $let\ \overline{X} = \overline{a}\ in\ t' \to^{l\,*} t' \equiv t$ by zero or more steps of (Elim), so $e \to^{l\,*} let\ \overline{X} = \overline{a}\ in\ t' \to^{l\,*} t$, that is $e \to^{l\,*} t$.

Finally, parts *ii)* and *iv)* follow from *ii)*, *iii)* and Theorem 4. □

As an immediate corollary of this completeness result and soundness (Theorem 10), we obtain the following result relating let-rewriting to CRWL and CRWL$_{let}$ for total c-terms, which gives a clean and easy way to understand the formulation of the adequacy of let-rewriting.

Corollary 1 (*Equivalence of CRWL$_{let}$ and let-rewriting for total values*)
For any program $\mathcal{P}$, $e \in LExp$, and $t \in CTerm$ we have

$$\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t \text{ iff } e \to^{l\,*} t.$$

Besides if neither $\mathcal{P}$ nor e have lets then we also have

$$\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t \text{ iff } e \to^{l\,*} t.$$

As final consequence of Theorems 10 and 11 we obtain another strong equivalence result for both formalisms, this time expressed in terms of semantics and hypersemantics.

Theorem 12 (*Equivalence of CRWL$_{let}$ and let-rewriting*)
For any program $\mathcal{P}$ and $e \in LExp$:

i) $[\![e]\!] = \{|e'| \mid e \to^{l^*} e'\}\!\downarrow$
ii) $[\![\![e]\!]\!] = \lambda\theta \in CSubst_\perp.(\{|e'| \mid e \to^{l^*} e'\}\!\downarrow)$

where $\downarrow$ is the downward closure operator defined in Lemma 5.

Proof

i) We prove both inclusions. Regarding $[\![e]\!] \subseteq \{|e'| \mid e \to^{l^*} e'\}\!\downarrow$, assume $t \in [\![e]\!]$. By Theorem 11 there must exist some $e' \in LExp$ such that $e \to^{l^*} e'$ and $t \sqsubseteq |e'|$, therefore $|e'| \in \{|e'| \mid e \to^{l^*} e'\}$. But this, combined with $t \sqsubseteq |e'|$, results in $t \in \{|e'| \mid e \to^{l^*} e'\}\!\downarrow$.
Regarding the other inclusion, consider some $t \in \{|e'| \mid e \to^{l^*} e'\}\!\downarrow$. By definition of the $\downarrow$ operator, there must exist some $e' \in LExp$ such that $t \sqsubseteq |e'|$ and $e \to^{l^*} e'$. But that implies $|e'| \in [\![e]\!]$, by Theorem 10, which combined with $t \sqsubseteq |e'|$ and the polarity property (Proposition 3) gives us that $t \in [\![e]\!]$.
ii) Trivial by applying the previous item and the definition of hypersemantics of an expression.

□

5 Semantic reasoning

Having equivalent notions of semantics and reduction allows to reason interchangeably at the rewriting and semantic levels. In this section we show the power of such technique in different situations. We start with a concrete example, adapted from (López-Fraguas et al. 2009b), where semantic reasoning leads easily to conclusions non-trivially achievable when thinking directly in operational terms.

Example 4
Imagine a program using constructors $a, b \in CS^0, c \in CS^1, d \in CS^2$ and defining a function $f \in FS^1$ for which we know that $f(a)$ can be let-rewritten to $c(a)$ and $c(b)$ but no other c-terms. Consider also an expression e having $f(a)$ as subexpression, i.e., e has the shape $\mathcal{C}[f(a)]$. We are interested now in the following question: can we safely replace in e the subexpression $f(a)$ by any other ground expression e'

let-reducible to the same set of values[3]? By safely we mean not changing the values reachable from e.

The question is less trivial than it could appear. For instance, if reductions were made with term rewriting instead of let-rewriting —i.e., considering run-time instead of call-time choice— the answer is negative (López-Fraguas et al. 2009b). To see that, consider the program

$$\begin{array}{lll} f(a) \to c(a) & g \to a & h(c(X)) \to d(X,X) \\ f(a) \to c(b) & g \to b & \end{array}$$

and the expressions $e \equiv h(f(a))$ and $e' \equiv c(g)$. All this is compatible with the assumptions of our problem. However, e is reducible by term rewriting only to $d(a,a)$ and $d(b,b)$, while replacing $f(a)$ by e' in e gives $h(c(g))$, which is reducible by term rewriting to two additional values, $d(a,b)$ and $d(b,a)$; thus, the replacement of $f(a)$ by e' has been unsafe.

However, the answer to our question is affirmative in general for let-rewriting, as it is very easily proved by a semantic reasoning using compositionality of CRWL$_{let}$: the assumption on $f(a)$ and e' means that they have the same denotation $[\![f(a)]\!] = [\![e']\!] = \{c(a), c(b)\}\downarrow$ and, since they are ground, the same hyperdenotation $[\![\![f(a)]\!]\!] = [\![\![e']\!]\!] = \lambda\theta.\{c(a), c(b)\}\downarrow$. By compositionality of hypersemantics, $\mathcal{C}[f(a)]$ and $\mathcal{C}[e']$ have the same (hyper)denotation, too. By equivalence of CRWL$_{let}$ and let-rewriting this implies that both expressions reach the same value by let-rewriting.

Despite its simplicity, the example raises naturally interesting questions about replaceability, for which semantic methods could be simpler than direct reasonings about reduction sequences. This is connected to the *full abstraction* problem that we have investigated for run-time and call-time choice in (López-Fraguas et al. 2009b; López-Fraguas and Rodríguez-Hortalá 2010).

Semantic methods can be also used to prove the correctness of new operational rules not directly provided by our set of let-rewriting rules. Such rules can be useful for different purposes: to make computations simpler, for program transformations, to obtain new properties of the framework, ... Consider for instance the following generalization of the (LetIn) rule in Figure 5:

(CLetIn) $\mathcal{C}[e] \to^l let\ X = e\ in\ \mathcal{C}[X]$, if $BV(\mathcal{C}) \cap FV(e) = \emptyset$ and X is fresh

This rule allows to create let-bindings in more situations and to put them in outer positions than the original (LetIn) rule. If we have not considered it in the definition of let-rewriting is because it would destroy the strong termination property of Proposition 2, as it is easy to see. However, this rule may shorten derivations. For instance, the derivation in Example 2 could be shortened to:

[3] More precisely, to the same set of shells in the sense of Theorem 12 part i).

$$\begin{array}{ll}
\underline{heads(repeat(coin))} & (CLetIn)\\
\to^{l} let\ C = coin\ in\ heads(\underline{repeat(C)}) & (Fapp)\\
\to^{l} let\ C = coin\ in\ heads(C : \underline{repeat(C)}) & (Fapp)\\
\to^{l} let\ C = coin\ in\ \underline{heads(C : C : repeat(C))} & (CLetIn)\\
\to^{l} let\ C = coin\ in\ let\ X = repeat(C)\ in\ \underline{heads(C : C : X)} & (Fapp)\\
\to^{l} let\ C = coin\ in\ \underline{let\ X = repeat(C)\ in\ (C, C)} & (Elim)\\
\to^{l} let\ C = \underline{coin}\ in\ (C, C) & (Fapp)\\
\to^{l} \underline{let\ C = 0\ in\ (C, C)} & (Bind)\\
\to^{l} (0, 0) &
\end{array}$$

Reasoning the correctness of (CLetIn) rule is not difficult by means of semantic methods. We only need to prove that the rule preserves hypersemantics.

Lemma 9
If $BV(\mathcal{C}) \cap FV(e) = \emptyset$ and X is fresh, then $[\![\mathcal{C}[e]]\!] = [\![let\ X = e\ in\ \mathcal{C}[X]]\!]$.

Proof
Assume an arbitrary $\theta \in CSubst_{\perp}$:

$$\begin{array}{ll}
[\![let\ X = e\ in\ \mathcal{C}[X]]\!]\theta = [\![(let\ X = e\ in\ \mathcal{C}[X])\theta]\!] & \\
= [\![let\ X = e\theta\ in\ \mathcal{C}\theta[X]]\!] & \text{as } X \text{ is fresh}\\
= \bigcup\limits_{t \in [\![e\theta]\!]} [\![(\mathcal{C}\theta[X])[X/t]]\!] & \text{by Theorem 5}\\
= \bigcup\limits_{t \in [\![e\theta]\!]} [\![\mathcal{C}\theta[t]]\!] & \text{as } X \text{ is fresh}\\
= [\![\mathcal{C}\theta[e\theta]]\!] & \text{by Theorem 5}\\
= [\![(\mathcal{C}[e])\theta]\!] = [\![\mathcal{C}[e]]\!]\theta &
\end{array}$$

□

The rule (CLetIn) is indeed used in some of the proofs in the online appendix, together with another derived rule:

$$\begin{array}{ll}
\textbf{(Dist)} & \mathcal{C}[let\ X = e_1\ in\ e_2] \to^{l} let\ X = e_1\ in\ \mathcal{C}[e_2],\\
& \text{if } BV(\mathcal{C}) \cap FV(e_1) = \emptyset \text{ and } X \notin FV(\mathcal{C})
\end{array}$$

which also preserves hypersemantics:

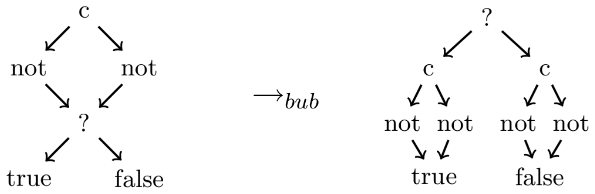
Lemma 10
If $BV(\mathcal{C}) \cap FV(e_1) = \emptyset$ and $X \notin FV(\mathcal{C})$ then $[\![\mathcal{C}[let\ X = e_1\ in\ e_2]]\!] = [\![let\ X = e_1\ in\ \mathcal{C}[e_2]]\!]$.

These ideas can be made more general. Consider the equivalence relation $e_1 \asymp e_2$ iff $[\![e_1]\!] = [\![e_2]\!]$. This relation is especially relevant because $e_1 \asymp e_2$ iff $\forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$, by Theorem 6. We can contemplate $\asymp$ as an abstract, although non-effective, reduction relation, of which the relations $\to^{lnf}$ of Section 4 and the rules (CLetIn) and (Dist) are particular subrelations. It is trivial to check that, by construction, the combined relation $\to^{l} \cup \asymp$ is sound and complete wrt. CRWL$_{let}$. We can use that relation to reason about the meaning or equivalence of let-expressions and programs. We could also employ it in the definition of on-demand evaluation strategies for let-rewriting. As any subrelation of $\to^{l} \cup \asymp$ is sound wrt. CRWL$_{let}$, an approach to strategies for let-rewriting could consist in defining a suitable operationally effective subrelation of $\rightarrow^l \cup \asymp$ and then proving its completeness and optimality (if it is the case).

5.1 A case study: correctness of bubbling

We develop here another nice application of the 'semantic route', where let-rewriting provides a good level of abstraction to formulate a new operational rule —*bubbling*— while the semantic point of view is appropriate for proving its correctness.

Bubbling, proposed in (Antoy et al. 2007), is an operational rule devised to improve the efficiency of functional logic computations. Its correctness was formally studied in (Antoy et al. 2006) in the framework of a variant (Echahed and Janodet 1998) of term graph rewriting. The idea of bubbling is to concentrate all nondeterminism of a system into a *choice* operation ? defined by the rules $X\ ?\ Y \rightarrow X$ and $X\ ?\ Y \rightarrow Y$, and to lift applications of ? out of their surrounding context, as illustrated by the following graph transformation taken from (Antoy et al. 2006):



As it is shown in (Antoy et al. 2007), bubbling can be implemented in such a way that many functional logic programs become more efficient, but we will not deal with these issues here.

Due to the technical particularities of term graph rewriting, not only the proof of correctness, but even the definition of bubbling in (Antoy et al. 2007; Antoy et al. 2006) are involved and need subtle care concerning the appropriate contexts over which choices can be bubbled. In contrast, bubbling can be expressed within our framework in a remarkably easy and abstract way as a new rewriting rule:

$$\textbf{(Bub)}\quad \mathcal{C}[e_1?e_2] \rightarrow^{bub} \mathcal{C}[e_1]?\mathcal{C}[e_2],\ \text{for } e_1, e_2 \in LExp$$

With this rule, the bubbling step corresponding to the graph transformation of the example above is:

$$\begin{array}{l} let\ X = true\ ?\ false\ in\ c(not(X), not(X)) \rightarrow^{bub} \\ let\ X = true\ in\ c(not(X), not(X))\ ?\ let\ X = false\ in\ c(not(X), not(X)) \end{array}$$

Notice that the effect of this bubbling step is not a shortening of any existing let-rewriting derivation; bubbling is indeed a genuine new rule, the correctness of which must be therefore subject of proof. Call-time choice is essential, since bubbling is not correct with respect to ordinary term rewriting, i.e., run-time choice.

Counterexample 2 (*Incorrectness of bubbling for run-time choice*)
Consider a function *pair* defined by the rule $pair(X) \rightarrow c(X, X)$ and the expression

$pair(0 \;?\; 1)$ for $c \in CS^2$ and $0, 1 \in CS^0$. Under term rewriting/run-time choice the derivation

$$pair(0 \;?\; 1) \to c(0 \;?\; 1, 0 \;?\; 1) \to c(0, 0 \;?\; 1) \to c(0, 1)$$

is valid. But if we performed the bubbling step

$$pair(0 \;?\; 1) \to^{bub} pair(0) \;?\; pair(1)$$

then the c-term $c(0, 1)$ would not be reachable anymore by term rewriting from $pair(0) \;?\; pair(1)$.

Formulating and proving the correctness of bubbling for call-time choice becomes easy by using semantics. As we did before, we simply prove that bubbling steps preserve hypersemantics. We need first a basic property of the (hyper)semantics of binary choice ?. Its proof stems almost immediately from the rules for ? and the definition of CRWL-(hyper)denotation.

Proposition 9 (*(Hyper)semantic properties of* ?)
For any $e_1, e_2 \in LExp_\perp$

i) $[\![e_1 \;?\; e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$
ii) $[\![\![e_1 \;?\; e_2]\!]\!] = [\![\![e_1]\!]\!] \Cup [\![\![e_2]\!]\!]$

Combining this property with some of the powerful hypersemantic results from Section 4.2 leads to an appealing proof of the correctness of bubbling.

Theorem 13 (*Correctness of bubbling for call-time choice*)
If $e \to^{bub} e'$ then $[\![\![e]\!]\!] = [\![\![e']\!]\!]$, for any $e, e' \in LExp$.

Proof
If $e \to^{bub} e'$ then $e = \mathcal{C}[e_1 \;?\; e_2]$ and $e' = \mathcal{C}[e_1] \;?\; \mathcal{C}[e_2]$, for some e_1, e_2. Then:

$$\begin{array}{ll}
[\![\![\mathcal{C}[e_1 \;?\; e_2]]\!]\!] = [\![\![\mathcal{C}]\!]\!][\![\![e_1 \;?\; e_2]\!]\!] & \text{by Theorem 6} \\
= [\![\![\mathcal{C}]\!]\!]([\![\![e_1]\!]\!] \Cup [\![\![e_2]\!]\!]) & \text{by Proposition 9 } \textit{ii)} \\
= [\![\![\mathcal{C}]\!]\!][\![\![e_1]\!]\!] \Cup [\![\![\mathcal{C}]\!]\!][\![\![e_2]\!]\!] & \text{by Proposition 7} \\
= [\![\![\mathcal{C}[e_1]]\!]\!] \Cup [\![\![\mathcal{C}[e_2]]\!]\!] & \text{by Theorem 6} \\
= [\![\![\mathcal{C}[e_1] \;?\; \mathcal{C}[e_2]]\!]\!] & \text{by Proposition 9 } \textit{ii)}
\end{array}$$

□

This property was proved also for the HO case in (López-Fraguas et al. 2008). But the proof given here is much more elegant thanks to the new semantic tools developed in Section 4.2.

6 Let-narrowing

It is well known that there are situations in functional logic computations where rewriting is not enough and must be lifted to some kind of *narrowing*, because the expression being reduced contains variables for which different bindings might produce different evaluation results. This might happen either because variables are already present in the initial expression to reduce, or due to the presence of extra variables in the program rules. In the latter case let-rewriting certainly works, but not in an effective way, since the parameter passing substitution in the rule (Fapp) of Figure 5 (page 16) 'magically' guesses the appropriate values for those extra variables (see Example 6 below). Some works (Antoy and Hanus 2006; Dios-Castro and López-Fraguas 2007; Braßel and Huch 2007) have proved that guessing can be replaced by a systematic non-deterministic generation of all (ground) possible values. However, this does not cover all aspects of narrowing, which is able to produce non-ground answers, while generator functions are not. In this section we present *let-narrowing*, a natural lifting of let-rewriting devised to effectively deal with free and extra variables.

Using the notation of contexts, the standard definition of narrowing as a lifting of term rewriting in ordinary TRS's is the following: $\mathcal{C}[f(\overline{t})] \leadsto_\theta \mathcal{C}\theta[r\theta]$, if θ is a mgu of $f(\overline{t})$ and $f(\overline{s})$, where $f(\overline{s}) \to r$ is a fresh variant of a rule of the TRS. The requirement that the binding substitution θ is a mgu can be relaxed to accomplish with certain narrowing strategies like needed narrowing (Antoy et al. 2000), which use unifiers but not necessarily most general ones.

This definition of narrowing cannot be directly translated as it is to the case of let-rewriting, for two reasons. First, binding substitutions must be c-substitutions, as for the case of let-rewriting. Second, let-bound variables should not be narrowed, but their values should be rather obtained by evaluation of their binding expressions. The following example illustrates some of the points above.

Example 5

Consider the following program over Peano natural numbers:

$$\begin{array}{ll}
0 + Y \to Y & \mathit{even}(X) \to \mathit{if}\ (Y{+}Y == X)\ \mathit{then}\ \mathit{true} \\
s(X) + Y \to s(X + Y) & \mathit{if}\ \mathit{true}\ \mathit{then}\ Y \to Y \\
0 == 0 \to \mathit{true} & s(X) == s(Y) \to X == Y \\
0 == s(Y) \to \mathit{false} & s(X) == 0 \to \mathit{false} \\
\mathit{coin} \to 0 & \mathit{coin} \to s(0)
\end{array}$$

Notice the extra variable Y in the rule for *even*. The evaluation of *even(coin)* by let-rewriting could start as follows:

$$\begin{array}{l}
\mathit{even}(\mathit{coin}) \to^l \mathit{let}\ X = \mathit{coin}\ \mathit{in}\ \mathit{even}(X) \\
\to^l \mathit{let}\ X = \mathit{coin}\ \mathit{in}\ \mathit{if}\ (Y + Y == X)\ \mathit{then}\ \mathit{true} \\
\to^{l\,*} \mathit{let}\ X = \mathit{coin}\ \mathit{in}\ \mathit{let}\ U = Y + Y\ \mathit{in}\ \mathit{let}\ V = (U == X)\ \mathit{in}\ \mathit{if}\ V\ \mathit{then}\ \mathit{true} \\
\to^{l\,*} \mathit{let}\ U = Y + Y\ \mathit{in}\ \mathit{let}\ V = (U == 0)\ \mathit{in}\ \mathit{if}\ V\ \mathit{then}\ \mathit{true}
\end{array}$$

Now, because all function applications involve variables, the evaluation cannot continue merely by rewriting, and therefore narrowing is required instead. We should not perform standard narrowing steps that bind already let-bound variables; otherwise, the syntax of let-expressions can be lost. For instance, narrowing at *if V then true* generates the binding $[V/\mathit{true}]$ that, if applied naively to the surrounding context, results in the syntactically illegal expression:

$$\mathit{let}\ U{=}Y{+}Y\ \mathit{in}\ \mathit{let}\ \mathit{true}{=}(U{=}{=}0)\ \mathit{in}\ \mathit{true}$$

(X) $e \rightsquigarrow^{l}_{\epsilon} e'$ if $e \rightarrow^{l} e'$ using $\boldsymbol{X} \in \{LetIn, Bind, Elim, Flat\}$ in Figure 5 (page 16).

(Narr) $f(\overline{t}) \rightsquigarrow^{l}_{\theta} r\theta$, for any fresh variant $(f(\overline{p}) \rightarrow r) \in \mathcal{P}$ and $\theta \in CSubst$ such that $f(\overline{t})\theta \equiv f(\overline{p})\theta$.

(Contx) $\mathcal{C}[e] \rightsquigarrow^{l}_{\theta} \mathcal{C}\theta[e']$, for $\mathcal{C} \neq []$, if $e \rightsquigarrow^{l}_{\theta} e'$ by any of the previous rules, and if the step is (Narr) using $(f(\overline{p}) \rightarrow r) \in \mathcal{P}$, then:

(i) $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$

(ii) $vRan(\theta|_{\setminus var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$

Fig. 6. Rules of the let-narrowing relation $\rightsquigarrow^{l}$

What is harmless is to perform narrowing at $Y + Y$ (Y is a free variable). This gives the substitution $[Y/0]$ and the result 0 for the subexpression $Y + Y$. Placing it in its surrounding context, the derivation continues as follows:

$$\begin{array}{l} let\ U = 0\ in\ let\ V = (U == 0)\ in\ if\ V\ then\ true \\ \rightarrow^{l} let\ V = (0 == 0)\ in\ if\ V\ then\ true \\ \rightarrow^{l} let\ V = true\ in\ if\ V\ then\ true \\ \rightarrow^{l} if\ true\ then\ true \rightarrow^{l} true \end{array}$$

The previous example shows that let-narrowing *must protect bound variables* against substitutions, which is the key observation for defining narrowing in presence of let-bindings.

The one-step let-narrowing relation $e \rightsquigarrow^{l}_{\theta} e'$ (assuming a given program $\mathcal{P}$) is defined in Figure 6.

- The rule **(X)** collects *(Elim), (Bind), (Flat), (LetIn)* of $\rightarrow^{l}$, that remain the same in $\rightsquigarrow^{l}$, except for the decoration with the empty substitution ϵ.
- The rule **(Narr)** performs a narrowing step in a proper sense. To avoid unnecessary loss of generality or applicability of our approach, we do not impose θ to be a mgu. For the sake of readability, we will sometimes decorate (Narr) steps with $\theta|_{FV(f(\overline{t}))}$ instead of θ, i.e., with the projection over the variables in the narrowed expression.
- The rule **(Contx)** indicates how to use the previous rules in inner positions. The condition $\mathcal{C} \neq [\ ]$ simply avoids trivial overlappings of (Contx) with the previous rules. The rest of the conditions are set to ensure that the combination of (Contx) with (Narr) makes a proper treatment of bound variables:

  — *(i)* expresses the protection of bound variables against narrowing justified in Example 5.

  — *(ii)* is a rather technical condition needed to prevent undesired situations when the narrowing step has used a program rule with extra variables and a unifier θ which is not a mgu. Concretely, the condition states that the bindings created by θ for the extra variables in the program rule do not introduce variables that are bound by the surrounding context $\mathcal{C}$. To see the problems that can arise without *(ii)*, consider for instance the program rules $f \rightarrow Y$ and $loop \rightarrow loop$ and the expression $let\ X = loop\ in\ f$. A legal reduction for this expression, respecting condition *(ii)* could be the following:

$$let\ X = loop\ in\ f\ \rightsquigarrow^{l}{}_{\epsilon}\ let\ X = loop\ in\ Z$$

by applying (Narr) to f with $\theta = \epsilon$ taking the fresh variant rule $f \to Z$, and using (Contx) for the whole expression. However, if we drop condition (ii) we could perform a similar derivation using the same fresh variant of the rule for f, but now using the substitution $\theta = [Z/X]$:

$$let\ X = loop\ in\ f\ \rightsquigarrow^{l}{}_{\epsilon}\ let\ X = loop\ in\ X$$

which is certainly not intended because the free variable Z in the previous derivation appears now as a bound variable, i.e., we get an undesired capture of variables.
We remark that if the substitution θ in (Narr) is chosen to be a standard mgu[4] of $f(\bar{t})$ and $f(\overline{p})$ (which is always possible) then the condition *(ii)* is always fulfilled.

The one-step relation $\rightsquigarrow^{l}{}_{\theta}$ is extended in the natural way to the multiple-steps narrowing relation $\rightsquigarrow^{l^*}_{\theta}$, which is defined as the least relation verifying:

$$e \rightsquigarrow^{l^*}_{\epsilon} e \qquad e \rightsquigarrow^{l}{}_{\theta_1} e_1 \rightsquigarrow^{l}{}_{\theta_2} \ldots e_n \rightsquigarrow^{l}{}_{\theta_n} e' \ \Rightarrow\ e \rightsquigarrow^{l^*}_{\theta_1 \ldots \theta_n} e'$$

We write $e \rightsquigarrow^{l^n}_{\theta} e'$ for a n-steps narrowing sequence.

Example 6
Example 5 essentially contains already a narrowing derivation. For the sake of clarity, we repeat it here making explicit the rule of let-narrowing used at each step (maybe in combination with (Contx), which is not written). Besides, if the step uses (Narr), the narrowed expression is underlined.

$$\begin{array}{ll}
even(coin) \rightsquigarrow^{l}{}_{\epsilon} & (LetIn) \\
let\ X = coin\ in\ \underline{even(X)} \rightsquigarrow^{l}{}_{\epsilon} & (Narr) \\
let\ X = coin\ in\ if\ Y+Y == X\ then\ true \rightsquigarrow^{l^3}_{\epsilon} & (LetIn^2, Flat) \\
let\ X = \underline{coin}\ in\ let\ U = Y+Y\ in & \\
\qquad let\ V = (U == X)\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Narr) \\
let\ X = 0\ in\ let\ U = Y+Y\ in & \\
\qquad let\ V = (U == X)\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Bind) \\
let\ U = \underline{Y+Y}\ in\ let\ V = (U == 0)\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{[Y/0]} & (Narr) \\
let\ U = 0\ in\ let\ V = (U == 0)\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Bind) \\
let\ V = \underline{(0 == 0)}\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Narr) \\
let\ V = true\ in\ if\ V\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Bind) \\
if\ true\ then\ true \rightsquigarrow^{l}{}_{\epsilon} & (Narr) \\
true &
\end{array}$$

Notice that all (Narr) steps in the derivation except one have ϵ as narrowing substitution (because of the projection over the variables of the narrowed expression), so they are really rewriting steps. An additional remark that could help to

[4] By standard mgu of t, s we mean an idempotent mgu θ with $dom(\theta) \cup ran(\theta) \subseteq var(t) \cup var(s)$.

further explain the relationship between the let-narrowing relation $\leadsto^l$ and the let-rewriting relation $\to^l$ is the following: since we have $even(coin) \leadsto^l{}_\theta\ true$ for some θ, but $even(coin)$ is ground, Theorem 14 in next section ensures that there must be also a successful let-rewriting derivation $even(coin) \to^{l\,*}\ true$. This derivation could have the form:

$$\begin{array}{ll} even(coin) \to^l & (LetIn) \\ let\ X = coin\ in\ even(X) \to^l & (Fapp) \\ let\ X = coin\ in\ if\ (0+0 == X)\ then\ true \to^l & \\ \ldots\ldots\ldots \ \to^l\ true & \end{array}$$

The indicated (Fapp)-step in this let-rewriting derivation has used the substitution $[Y/0]$, thus anticipating and 'magically guessing' the correct value of the extra variable Y of the rule of $even$. In contrast, in the let-narrowing derivation the binding for Y is not done while reducing $even(X)$ but in a later (Narr)-step over $Y + Y$. This corresponds closely to the behavior of narrowing-based systems like Toy or Curry.

6.1 Soundness and completeness of the let-narrowing relation $\leadsto^l$

In this section we show the adequacy of let-narrowing wrt. let-rewriting. From now on we assume a fixed program $\mathcal{P}$.

As usual with narrowing relations, soundness results are not difficult to formulate and prove. The following *soundness* result for $\leadsto^l$ states that we can mimic any $\leadsto^l$ derivation with $\to^l$ by applying over the starting expression the substitution computed by the original let-narrowing derivation.

Theorem 14 (*Soundness of the let-narrowing relation* $\leadsto^l$)
For any $e, e' \in LExp$, $e \leadsto^{l^*}_\theta e'$ implies $e\theta \to^{l\,*} e'$.

Completeness is more complicated to prove. The key result is a generalization to let-rewriting of Hullot's *lifting lemma* (Hullot 1980) for classical term rewriting and narrowing. It states that any rewrite sequence for a particular instance of an expression can be generalized by a narrowing derivation.

Lemma 11 (*Lifting lemma for the let-rewriting relation* $\to^l$)
Let $e, e' \in LExp$ such that $e\theta \to^{l^*} e'$ for some $\theta \in CSubst$, and let $\mathcal{W}, \mathcal{B} \subseteq \mathcal{V}$ with $dom(\theta) \cup FV(e) \subseteq \mathcal{W}$, $BV(e) \subseteq \mathcal{B}$ and $(dom(\theta) \cup vran(\theta)) \cap \mathcal{B} = \emptyset$, and for each (Fapp) step of $e\theta \to^{l^*} e'$ using a rule $R \in \mathcal{P}$ and a substitution $\gamma \in CSubst$ then $vran(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$. Then there exist a derivation $e \leadsto^{l^*}_\sigma e''$ and $\theta' \in CSubst$ such that:

$$\text{(i) } e''\theta' = e' \qquad \text{(ii) } \sigma\theta' = \theta[\mathcal{W}] \qquad \text{(iii) } (dom(\theta') \cup vran(\theta')) \cap \mathcal{B} = \emptyset$$

Besides, the let-narrowing derivation can be chosen to use mgu's at each (Narr) step. Graphically:

$$\begin{tikzcd}
e \arrow[r, squiggly, dotted, "l^*", "\sigma"'] \arrow[d, mapsto, "\theta"'] & e'' \arrow[d, mapsto, dashed, "\theta'"] \\
e\theta \arrow[r, "l^*"] & e'
\end{tikzcd}$$

With the aid of this lemma we are now ready to state and prove the following strong completeness result for $\leadsto^{l}$.

Theorem 15 (*Completeness of the let-narrowing relation* $\leadsto^{l}$)
Let $e, e' \in LExp$ and $\theta \in CSubst$. If $e\theta \rightarrow^{l}{}^{*} e'$, then there exist a let-narrowing derivation $e \leadsto^{l^*}_{\sigma} e''$ and $\theta' \in CSubst$ such that $e''\theta' \equiv e'$ and $\sigma\theta' = \theta[FV(e)]$.

Proof
Applying Lemma 11 to $e\theta|_{FV(e)} \rightarrow^{l}{}^{*} e'$ with $\mathcal{W} = FV(e)$ and $\mathcal{B} = BV(e)$, as $e\theta|_{FV(e)} \equiv e\theta$ and the additional conditions over $\mathcal{B}$ hold by the variable convention.
□

Finally, by combining Theorems 14 and 15, we obtain a strong adequacy theorem for let-narrowing with respect to let-rewriting.

Theorem 16 (*Adequacy of the let-narrowing relation* $\leadsto^{l}$ *wrt.* $\rightarrow^{l}$)
Let $e, e_1 \in LExp$ and $\theta \in CSubst$, then:

$$e\theta \rightarrow^{l}{}^{*} e_1 \;\Leftrightarrow\; \begin{array}{l} \text{there exist a let-narrowing derivation } e \leadsto^{l^*}_{\sigma} e_2 \text{ and} \\ \text{some } \theta' \in CSubst \text{ such that } \sigma\theta' = \theta[FV(e)],\; e_2\theta' \equiv e_1 \end{array}$$

Proof
($\Rightarrow$) Assume $e\theta \rightarrow^{l^*} e_1$. As $e\theta|_{FV(e)} \equiv e\theta$ then trivially $e\theta|_{FV(e)} \rightarrow^{l}{}^{*} e_1$. We can apply Lemma 11 taking $\mathcal{W} = FV(e)$ to get $e \leadsto^{l^*}_{\sigma} e_2$ such that there exists $\theta' \in CSubst$ with $\sigma\theta' = \theta|_{FV(e)}[\mathcal{W}]$ and $e_2\theta' \equiv e_1$. But as $\mathcal{W} = FV(e)$ then $\sigma\theta' = \theta|_{FV(e)}[\mathcal{W}]$ implies $\sigma\theta' = \theta[FV(e)]$.
We remark that the lifting lemma ensures that the narrowing derivation can be chosen to use mgu's at each (Narr) step.
($\Leftarrow$) Assume $e \leadsto^{l^*}_{\sigma} e_2$ and θ' under the conditions above. Then by Theorem 14 we have $e\sigma \rightarrow^{l}{}^{*} e_2$. As $\rightarrow^{l}$ is closed under c-substitutions (Lemma 2) then $e\sigma\theta' \rightarrow^{l}{}^{*} e_2\theta'$. But as $\sigma\theta' = \theta[FV(e)]$, then $e\theta \equiv e\sigma\theta' \rightarrow^{l}{}^{*} e_2\theta' \equiv e_1$.
□

6.2 Organizing computations

Deliberately, in this paper we have kept the definitions of let-rewriting and narrowing apart from any particular computation strategy. In this section we explain rather informally how the ideas of some known strategies for functional logic programming (Antoy 2005) can be adapted also to our formal setting. For the sake of brevity we focus only on let-narrowing computations. As a running example, consider the program

$$\begin{array}{ll} leq(0,Y) \rightarrow true & f(0) \rightarrow 0 \\ leq(s(X),0) \rightarrow false & \\ leq(s(X),s(Y)) \rightarrow leq(X,Y) & \end{array}$$

and the initial expression $leq(X, f(Y))$ to be let-narrowed using it.

As a first remark, when designing a strategy one can freely use 'peeling' steps in a *don't care* manner using the relation $\rightarrow^{lnf}$ (Definition 3), since it is terminating and (hyper-)semantics-preserving. In our case one step suffices: $leq(X, f(Y)) \rightsquigarrow^{l}{}_{\epsilon} let\ U = f(Y)\ in\ leq(X,U)$. After a peeling (multi-)step, a (Narr) step must be done. Where? Certainly, the body $leq(\ldots)$ must be narrowed at some point. One *don't know* possibility is narrowing at $leq(X,U)$ using the first rule for leq that does not bind U: $let\ U = f(Y)\ in\ leq(X,U) \rightsquigarrow^{l}{}_{[X/0]} let\ U = f(Y)\ in\ true$. A new peeling step leads to a first final result $true$, with computed substitution $[X/0]$.

The second and third rules for leq could lead to more results. Those rules have non-variable patterns as second arguments, and then the bound variable U in $leq(X,U)$ inhibits a direct (Narr) step in that position. Typically it is said that U is *demanded* by those leq rules. Therefore, we narrow $f(Y)$ to get values for U, and then we 'peel':

$$let\ U = f(Y)\ in\ leq(X,U) \rightsquigarrow^{l}{}_{[Y/0]} let\ U = 0\ in\ leq(X,U) \rightsquigarrow^{l}{}_{\epsilon} leq(X,0) \qquad (1)$$

The computation proceeds now by two don't know choices using the rules for leq, leading to two more solutions $(true, [Y/0, X/0])$ and $(false, [Y/0, X/s(Z)])$.

This implicitly applied strategy can be seen as a translation to let-narrowing of *lazy narrowing* (Moreno-Navarro and Rodríguez-Artalejo 1992; Alpuente et al. 2003). As a known drawback of lazy narrowing, notice that the second solution $(true, [Y/0, X/0])$ is redundant, since it is less general than the first one $(true, [X/0])$. Redundancy is explained because we have narrowed the expression $f(Y)$ whose evaluation was demanded only by some of the rules for the outer function application $leq(X, f(Y))$, but after that we have used the rules not demanding the evaluation (the first rule for leq). This problem is tackled successfully by *needed narrowing* (Antoy et al. 2000) which takes into account, when narrowing an inner expression, what are the rules for an outer function application demanding such evaluation. A needed narrowing step 'anticipates' the substitution that will perform these rules when they are to be applied. The ideas of needed narrowing can be adapted to our setting. In our example, we get the following derivation instead of (1):

$$\begin{array}{l} let\ U = f(Y)\ in\ leq(X,U) \rightsquigarrow^{l}{}_{[X/s(Z),Y/0]} let\ U = 0\ in\ leq(s(Z),U) \rightsquigarrow^{l}{}_{\epsilon} \\ leq(s(Z),0) \rightsquigarrow^{l}{}_{\epsilon} false \end{array} \qquad (1')$$

The first step does not use a mgu. This a typical feature of needed narrowing, and is also allowed by let-narrowing steps. Needed narrowing steps rely on *definitional trees* that structure demandness information from the rules of a given function. This information can be embedded also into a program transformation. There are simple transformations for which the transformed program, under a lazy narrowing regime using mgu's, obtains the same solutions than the original program (Zartmann 1997), although it is not guaranteed that the number of steps is also preserved. In our example, the definition of leq can be transformed as follows:

$$
\begin{array}{ll}
leq(0,Y) \to true & leqS(X,0) \to false \\
leq(s(X),Y) \to leqS(X,Y) & leqS(X,s(Y)) \to leq(X,Y)
\end{array}
$$

As happened with (1'), the derivation

$$
\begin{array}{l}
let\ U = f(Y)\ in\ leq(X,U) \ \rightsquigarrow^{l}{}_{[X/s(Z)]} let\ U = f(Y)\ in\ leqS(Z,U) \ \rightsquigarrow^{l}{}_{[Y/0]} \\
let\ U = 0\ in\ leqS(Z,U) \ \rightsquigarrow^{l}{}_{\epsilon} leqS(Z,0) \ \rightsquigarrow^{l}{}_{\epsilon} false
\end{array}
$$

gets rid of redundant solutions.

To which extent do our results guarantee the adequateness of the adaptation to let-narrowing of these strategies or others that could be defined? Certainly any strategy is *sound* for call-time choice semantics, because unrestricted $\rightsquigarrow^l$ is already sound (Theorem 14). This will be true also if the strategy uses derived rules in the sense of Section 5. With respect to completeness, we know that the space of let-narrowing derivations is complete wrt. let-rewriting (Theorem 15). But this does not imply the completeness of the strategy, which in general will determine a smaller narrowing space. Therefore completeness of the strategy must be proved independently. Such a proof may use semantic methods (i.e., prove completeness wrt. CRWL-semantics) or operational methods (i.e., prove completeness wrt. $\to^l$-derivations). We will not go deeper into the issue of strategies.

7 Let-rewriting versus classical term rewriting

In this section we examine the relationship between let-rewriting and ordinary term rewriting, with the focus put in the set of c-terms reachable by rewriting with each of these relations. As term rewriting is not able to handle expressions with let-bindings, during this section we assume that all considered programs do not have let-bindings in the right-hand side of its rules.

We will first prove in Section 7.1 that let-rewriting is sound with respect to term rewriting, in the sense that any c-term that can be reached by a let-rewriting derivation from a given expression can also be reached by a term rewriting derivation starting from the same expression. As we know, completeness does not hold in general because run-time choice computes more values than call-time choice for arbitrary programs. However, we will be able to prove completeness of let-rewriting wrt. term rewriting for the class of *deterministic* programs, a notion close to confluence that will be defined in Section 7.2. Finally, we will conclude in Section 7.3 with a comparison between let-narrowing and narrowing, that will follow easily from the results in previous subsections and the adequacy of let-narrowing to let-rewriting.

Thanks to the strong equivalence between CRWL and let-rewriting we can choose the most appropriate point of view for each of the two goals (soundness and completeness): we will use let-rewriting for proving soundness, and CRWL for defining the property of determinism and proving that, under determinism, completeness of let-rewriting wrt. term rewriting also holds.

7.1 Soundness of let-rewriting wrt. classical term rewriting

In order to relate let-rewriting to term rewriting, we first need to find a way for term rewriting to cope with let-bindings, which are not supported by its syntax, that is only able to handle expressions from *Exp*. Therefore, we define the following syntactic transformation from *LExp* into *Exp* that takes care of removing the let constructions, thus losing the sharing information they provide.

Definition 9 (*Let-binding elimination transformation*)
Given $e \in LExp$ we define its transformation into a let-free expression $\widehat{e} \in Exp$ as:

$$\begin{array}{l} \widehat{X} =_{def} X \\ \widehat{h(e_1,\ldots,e_n)} =_{def} h(\widehat{e_1},\ldots,\widehat{e_n}) \\ \widehat{let\ X = e_1\ in\ e_2} =_{def} \widehat{e_2}[X/\widehat{e_1}] \end{array}$$

Note that $\widehat{e} \equiv e$ for any $e \in Exp$.

We will need also the following auxiliary lemma showing the interaction between term rewriting derivations and substitution application.

Lemma 12 (*Copy lemma*)
For all $e, e_1, e_2 \in Exp$, $X \in \mathcal{V}$:

i) $e_1 \to e_2$ implies $e[X/e_1] \to^* e[X/e_2]$.
ii) $e_1 \to^* e_2$ implies $e[X/e_1] \to^* e[X/e_2]$.

Note how in *i)*, each of the different copies of e_1 introduced in e by the substitution has to be reduced to e_2 in a different term rewriting step in order to reach the expression $e[X/e_2]$.

Using this lemma we can get a first soundness result stating that the result of one let-rewriting step can also be obtained in zero or more steps of ordinary rewriting, after erasing the sharing information by means of the let-binding elimination transformation.

Lemma 13 (*One-Step Soundness of let-rewriting wrt. term rewriting*)
For all $e, e' \in LExp$ we have that $e \to^l e'$ implies $\widehat{e} \to^* \widehat{e'}$.

The remaining soundness results follow easily from this lemma. The first one shows how we can mimic let-rewriting with term rewriting through the let-binding elimination transformation. But then, as $\widehat{e} \equiv e$ for any $e \in Exp$, we conclude that for let-free expressions let-rewriting is a subrelation of term rewriting.

Theorem 17 (*Soundness of let-rewriting wrt. term rewriting*)
For any $e, e' \in LExp$ we have that $e \to^{l^*} e'$ implies $\widehat{e} \to^* \widehat{e'}$. As a consequence, if $e, e' \in Exp$ then $e \to^{l^*} e'$ implies $e \to^* e'$, i.e., $(\to^{l^*} \cap (Exp \times Exp)) \subseteq \to^*$.

Proof
The first part follows from an immediate induction on the length of the let-derivation, using Lemma 13 for the inductive step. The rest is obvious taking into account that $e \equiv \widehat{e}$ and $e' \equiv \widehat{e'}$ when $e, e' \in Exp$. □

To conclude this part, we can combine this last result with the equivalence of CRWL and let-rewriting, thus getting the following soundness result for CRWL with respect to term rewriting.

Theorem 18 (*Soundness of CRWL wrt. term rewriting*)
For any $e \in Exp$, $t \in CTerm_\perp$, if $e \rightarrowtriangle t$ then there exists $e' \in Exp$ such that $e \to^* e'$ and $t \sqsubseteq |e'|$.

Proof
Assume $e \rightarrowtriangle t$. By Theorem 11, there exists $e'' \in LExp$ such that $e \to^{l^*} e''$ and $t \sqsubseteq |e''|$. Then, by Theorem 17, we have $\widehat{e} \to^* \widehat{e''}$. As $e \in Exp$, we have $e \equiv \widehat{e}$ and we can choose $e' \equiv \widehat{e''} \in Exp$ so we get $e \to^* e'$. It is easy to check that $|e''| = |\widehat{e''}|$ and then we have $t \sqsubseteq |e''| = |\widehat{e''}| = |e'|$. □

7.2 Completeness of CRWL wrt. classical term rewriting

We prove here the completeness of the CRWL framework wrt. term rewriting for the class of CRWL-deterministic programs, which are defined as follows.

Definition 10 (*CRWL-deterministic program*)
A program $\mathcal{P}$ is *CRWL-deterministic* iff for any expression $e \in Exp_\perp$ its denotation $\llbracket e \rrbracket^{\mathcal{P}}$ is a directed set. In other words, iff for all $e \in Exp_\perp$ and $t_1, t_2 \in \llbracket e \rrbracket^{\mathcal{P}}$, there exists $t_3 \in \llbracket e \rrbracket^{\mathcal{P}}$ with $t_1 \sqsubseteq t_3$ and $t_2 \sqsubseteq t_3$.

Thanks to the equivalence of CRWL and let-rewriting, it is easy to characterize CRWL-determinism also in terms of let-rewriting derivations.

Lemma 14
A program $\mathcal{P}$ is CRWL-deterministic iff for any $e \in Exp$, $e', e'' \in LExp$ with $\mathcal{P} \vdash e \to^{l^*} e'$ and $\mathcal{P} \vdash e \to^{l^*} e''$ there exists $e''' \in LExp$ such that $\mathcal{P} \vdash e \to^{l^*} e'''$ and $|e'''| \sqsupseteq |e'|, |e'''| \sqsupseteq |e''|$.

Proof
For the left to right implication, assume a CRWL-deterministic program $\mathcal{P}$ and $e \in Exp$, $e', e'' \in LExp$ with $e \to^{l^*} e'$ and $e \to^{l^*} e''$. By part *iii*) of Theorem 10 we have $|e'|, |e''| \in \llbracket e \rrbracket$ and then by Definition 10 there exists $t \in \llbracket e \rrbracket$ such that $|e'|, |e''| \sqsubseteq t$. Now, by part *iii*) of Theorem 11 there exists $e''' \in LExp$ such that $e \to^{l^*} e'''$ and $t \sqsubseteq |e'''|$, so we have $|e'|, |e''| \sqsubseteq t \sqsubseteq |e'''|$ as expected.

Regarding the converse implication, assume $e \in Exp$ with $t_1, t_2 \in \llbracket e \rrbracket$. By part *iii*) of Theorem 11 there exist $e', e'' \in LExp$ such that $e \to^{l^*} e'$, $e \to^{l^*} e''$ and $t_1 \sqsubseteq |e'|$, $t_2 \sqsubseteq |e''|$. Then by hypothesis there exists $e''' \in LExp$ such that $e \to^{l^*} e'''$ and $|e'|, |e''| \sqsubseteq |e'''|$. Now, by part *iii*) of Theorem 10 we have $|e'''| \in \llbracket e \rrbracket$ and this $|e'''|$ is the t_3 of Definition 10 we are looking for, i.e., $t_3 \in \llbracket e \rrbracket$ and $t_1, t_2 \sqsubseteq t_3$. □

CRWL-determinism is intuitively close to confluence of term rewriting, but these two properties are not equivalent, as shown by the following example of a CRWL-deterministic but not confluent program.

Example 7
Consider the program $\mathcal{P}$ given by the rules

$$f \to a \quad f \to loop \quad loop \to loop$$

where a is a constructor. It is clear that $\to_{\mathcal{P}}$ is not confluent (f can be reduced to a and *loop*, which cannot be joined into a common reduct), but it is CRWL-deterministic, since $[\![f]\!]^{\mathcal{P}} = \{\bot, a\}$, $[\![loop]\!]^{\mathcal{P}} = \{\bot\}$ and $[\![a]\!]^{\mathcal{P}} = \{\bot, a\}$, which are all directed sets.

We conjecture that the reverse implication is true, i.e., that confluence of term rewriting implies CRWL-determinism. Nevertheless, a precise proof for this fact seems surprisingly complicated and we have not yet completed it.

A key ingredient in our completeness proof is the notion of CRWL-denotation of a substitution, which is the set of c-substitutions whose range can be obtained by CRWL-reduction over the range of the starting expression.

Definition 11 (*CRWL-denotation for a substitution*)
Given a program $\mathcal{P}$, the CRWL-denotation of a $\sigma \in Subst_\bot$ is defined as:

$$[\![\sigma]\!]^{\mathcal{P}}_{CRWL} = \{\theta \in CSubst_\bot \mid \forall X \in \mathcal{V},\ \mathcal{P} \vdash_{CRWL} \sigma(X) \rightarrowtriangle \theta(X)\}$$

We will usually omit the subscript CRWL and/or the superscript $\mathcal{P}$ when implied by the context.

Any substitution θ in the denotation of some substitution σ contains less information than σ, because it only holds in its range a finite part of the possibly infinite denotation of the expressions in the range of σ. We formalize this property in the following result.

Proposition 10
For all $\sigma \in Subst_\bot$, $\theta \in [\![\sigma]\!]$, we have that $\theta \trianglelefteq \sigma$.

Besides, we will use the notion of deterministic substitution, which is a substitution with only deterministic expressions in its range.

Definition 12 (*Deterministic substitution*)
The set $DSubst_\bot$ of *deterministic substitutions* for a given program $\mathcal{P}$ is defined as

$$DSubst_\bot = \{\sigma \in Subst_\bot \mid \forall X \in dom(\sigma).[\![\sigma(X)]\!] \text{ is a directed set}\}$$

Then $CSubst_\bot \subseteq DSubst_\bot$, and under any program $\forall \sigma \in Subst_\bot.[\![\sigma]\!] \subseteq CSubst_\bot \subseteq DSubst_\bot$. Note that the determinism of substitutions depends on the program, which gives meaning to the functions in its range. Obviously if a program is deterministic then $Subst_\bot = DSubst_\bot$.

A good thing about deterministic substitutions is that their denotation is always a directed set.

Proposition 11
For all $\sigma \in DSusbt_\perp$, $[\![\sigma]\!]$ is a directed set.

But the fundamental property of deterministic substitutions is that, for any CRWL-statement starting from an instance of an expression that has been constructed using a deterministic substitution, there is another CRWL-statement to the same value from another instance of the same expression that now has been built using a c-substitution taken from the denotation of the starting substitution. This property is a direct consequence of Proposition 11.

Lemma 15
For all $\sigma \in DSusbt_\perp$, $e \in Exp_\perp, t \in CTerm_\perp$,

$$\text{if } e\sigma \rightarrowtriangle t \text{ then } \exists\theta \in [\![\sigma]\!] \text{ such that } e\theta \rightarrowtriangle t$$

Proof (sketch)
We proceed by a case distinction over e. If $e \equiv X \in dom(\sigma)$ then we have $e\sigma \equiv \sigma(X) \rightarrowtriangle t$, and we can define

$$\theta(Y) = \begin{cases} t & \text{if } Y \equiv X \\ \perp & \text{if } Y \in dom(\sigma) \setminus \{X\} \\ Y & \text{otherwise} \end{cases}$$

Then it is easy to check that $\theta \in [\![\sigma]\!]$ and besides $e\theta \equiv \theta(X) \equiv t \rightarrowtriangle t$ by Lemma 5, so we are done. If $e \equiv X \in \mathcal{V} \setminus dom(\sigma)$ then we have $e\sigma \equiv \sigma(X) \equiv X \rightarrowtriangle t$, and given $\overline{Y} = dom(\sigma)$ it is easy to check that $[\overline{Y/\perp}] \in [\![\sigma]\!]$, and besides $e[\overline{Y/\perp}] \equiv X \rightarrowtriangle t$ by hypothesis.

Finally if $e \notin \mathcal{V}$ we proceed by induction on the structure of the proof for $e\sigma \rightarrowtriangle t$. The interesting cases are those for (DC) and (OR) where we use that $\sigma \in DSusbt_\perp$, so by Proposition 11 its denotation is directed. Then there must exist some $\theta \in [\![\sigma]\!]$ which is greater than each of the θ_i obtained by induction hypothesis over the premises of the starting CRWL-proof for $e\sigma \rightarrowtriangle t$. Using the monotonicity of Proposition 5 we can prove $e\theta \rightarrowtriangle t$, which also holds for CRWL, by Theorem 4 (see Appendix A, page 88 for details). □

Now we are finally ready to prove our first completeness result of CRWL wrt. term rewriting, for deterministic programs.

Lemma 16 (*Completeness lemma for CRWL wrt. term rewriting*)
Let $\mathcal{P}$ be a CRWL-deterministic program, and $e, e' \in Exp$. Then:

$$e \to^* e' \text{ implies } [\![e']\!] \subseteq [\![e]\!]$$

Proof
We can just prove this result for $e \to e'$, then its extension for an arbitrary number of term rewriting steps holds by a simple induction on the length of the term rewriting derivation, using transitivity of $\subseteq$.

Assume $e \to e'$, then the step must be of the shape $e \equiv \mathcal{C}[f(\overline{p})\sigma] \to \mathcal{C}[r\sigma] \equiv e'$ for some program rule $(f(\overline{p}) \to r) \in \mathcal{P}$, $\sigma \in Subst$. First, let us focus on the case for

$\mathcal{C} = [\,]$, and then assume some $t \in CTerm_\perp$ such that $\mathcal{P} \vdash_{CRWL} e' \equiv r\sigma \to t$. As $\mathcal{P}$ is deterministic then $\sigma \in DSubst_\perp$, therefore by Lemma 15 there must exist some $\theta \in [\![\sigma]\!]$ such that $\mathcal{P} \vdash_{CRWL} r\theta \to t$. But then we can use θ to build the following CRWL-proof.

$$\frac{\ldots\ p_i\theta \to p_i\theta \ldots\ \ r\theta \to t}{f(\overline{p})\theta \to t}\ OR$$

where for each $p_i \in \overline{p}$ we have $\mathcal{P} \vdash_{CRWL} p_i\theta \to p_i\theta$ by Lemma 5, as $p_i \in CTerm$ because $\mathcal{P}$ is a constructor system, and so $p_i\theta \in CTerm_\perp$, as $\theta \in [\![\sigma]\!] \subseteq CSubst_\perp$. But we also have $\theta \trianglelefteq \sigma$ by Proposition 10, therefore by applying the monotonicity for substitutions from Proposition 5 —which also holds for CRWL, by Theorem 4— we get $\mathcal{P} \vdash_{CRWL} e \equiv f(\overline{p})\sigma \to t$. Hence $[\![e']\!] = [\![r\sigma]\!] \subseteq [\![f(\overline{p})\sigma]\!] = [\![e]\!]$.

Finally, we can generalize this result to arbitrary contexts by using the compositionality of CRWL from Theorem 1. Given a term rewriting step $e \equiv \mathcal{C}[f(\overline{p})\sigma] \to \mathcal{C}[r\sigma] \equiv e'$ then by the proof for $\mathcal{C} = [\,]$ we get $[\![r\sigma]\!] \subseteq [\![f(\overline{p})\sigma]\!]$, but then

$$\begin{array}{lll} & [\![e']\!] = [\![\mathcal{C}[r\sigma]]\!] & \\ = & \bigcup\limits_{t \in [\![r\sigma]\!]} [\![\mathcal{C}[t]]\!] & \text{by Theorem 1} \\ \subseteq & \bigcup\limits_{t \in [\![f(\overline{p})\sigma]\!]} [\![\mathcal{C}[t]]\!] & \text{as } [\![r\sigma]\!] \subseteq [\![f(\overline{p})\sigma]\!] \\ = & [\![\mathcal{C}[f(\overline{p})\sigma]]\!] = [\![e]\!] & \text{by Theorem 1} \end{array}$$

□

The previous lemma, together with the equivalence of CRWL and let-rewriting given by Theorem 12 and Theorem 4, allows us to obtain a strong relationships between term rewriting, let-rewriting and CRWL, for the class of CRWL-deterministic programs.

Theorem 19

Let $\mathcal{P}$ be a CRWL-deterministic program, and $e, e' \in Exp, t \in CTerm$. Then:

a) $e \to^* e'$ implies $e \to^{l^*} e''$ for some $e'' \in LExp$ with $|e''| \sqsupseteq |e'|$.

b) $e \to^* t$ iff $e \to^{l^*} t$ iff $\mathcal{P} \vdash_{CRWL} e \to t$.

Notice that in part *a)* we cannot ensure $e \to^* e'$ implies $e \to^{l^*} e'$, because term rewriting can reach some intermediate expressions not reachable by let-rewriting. For instance, given the deterministic program with the rules $g \to a$ and $f(x) \to c(x,x)$, we have $f(g) \to^* c(g,a)$, but $f(g) \not\to^{l^*} c(g,a)$. Still, parts *a)* is a strong completeness results for let-rewriting wrt. term rewriting for deterministic programs, since it says that the outer constructed part obtained in a rewriting derivation can be also obtained or even refined in a let-rewriting derivation. Combined with Theorem 17, part *a)* expresses a kind of equivalence between let-rewriting and term rewriting, valid for general derivations, even non-terminating ones. For derivations reaching a constructor term (not further reducible), part *b)* gives an even stronger equivalence result.

7.3 Let-narrowing versus narrowing for deterministic systems

Joining the results of the previous section with the adequacy of let-narrowing to let-rewriting, we can easily establish some relationships between let-narrowing and ordinary term rewriting/narrowing, summarized in the following result.

Theorem 20
For any program $\mathcal{P}$, $e \in Exp, \theta \in CSubst$ and $t \in CTerm$:

a) If $e \rightsquigarrow_\theta^{l^*} t$ then $e\theta \rightarrow^* t$.
b) If in addition $\mathcal{P}$ is CRWL-deterministic, then:

b$_1$) If $e\theta \rightarrow^* t$ then $\exists t' \in CTerm,\ \sigma, \theta' \in CSubst$ such that $e \rightsquigarrow_\sigma^{l^*} t'$, $t'\theta' \equiv t$ and $\sigma\theta' = \theta[var(e)]$.
b$_2$) If $e \rightsquigarrow_\theta^* t$, the same conclusion of *(b$_1$)* holds.

Part a) expresses soundness of $\rightsquigarrow^l$ wrt. term rewriting, and part b) is a completeness result for $\rightsquigarrow^l$ wrt. term rewriting/narrowing, for the class of deterministic programs.

Proof
Part a) follows from soundness of let-narrowing wrt. let-rewriting (Theorem 14) and soundness of let-rewriting wrt. term rewriting of Theorem 19.

For part b_1), for let-narrowing, assume $e\theta \rightarrow^* t$. By the completeness of let-rewriting wrt. term rewriting for deterministic programs (Theorem 19), we have $e\theta \rightarrow^{l\,*} t$, and then by the completeness of let-narrowing wrt. let-rewriting (Theorem 15), there exists a narrowing derivation $e \rightsquigarrow_\sigma^{l^*} t'$ with $t'\theta' = t$ and $\sigma\theta' = \theta[FV(e)]$. But notice that for $e \in Exp$, the sets $FV(e)$ and $var(e)$ coincide, and the proof is finished.

Finally, b_2) follows simply from soundness of (ordinary) narrowing wrt. term rewriting and b_1). □

8 Conclusions

This paper contains a thorough presentation of the theory of first order let-rewriting and let-narrowing for constructor-based term rewriting systems. These two relations are simple notions of one-step reduction that express sharing as it is required by the call-time choice semantics of non-determinism adopted in the functional logic programming paradigm. In a broad sense, let-rewriting and let-narrowing can be seen as particular syntactical presentations of term graph rewriting and narrowing. However, keeping our formalisms very close to the syntax and basic notions of term rewriting systems (terms, substitutions, syntactic unification,...) has been an essential aid in establishing strong equivalence results with respect to the CRWL-framework —a well-established realization of call-time choice semantics—, which was one of the main aims of the paper.

Along the way of proving such equivalence we have developed powerful semantic tools that are interesting in themselves. Most remarkably, the CRWL$_{let}$-logic, a conservative extension of CRWL that deals with let-bindings, and the notion of hypersemantics of expressions and contexts, for which we prove deep compositionality results not easily achievable by thinking directly in terms of reduction sequences.

We have shown in several places the methodological power of having provably equivalent reduction-based and logic-based semantics. In some occasions, we have used the properties of the CRWL-semantics to investigate interesting aspects of reductions, as replaceability conditions or derived operational rules, like bubbling. In others, we have followed the converse way. For instance, by transforming let-rewriting reductions into ordinary term rewriting reductions, we easily concluded that let-rewriting (call-time choice) provides less computed values than term rewriting (run-time choice). By using again semantic methods, we proved the opposite inclusion for deterministic programs, obtaining for such programs an equivalence result of let-rewriting and term rewriting.

In our opinion, the different pieces of this work can be used separately for different purposes. The CRWL$_{let}$-logic provides a denotational semantics reflecting call-time choice for programs making use of local bindings. The let-rewriting and let-narrowing relations provide clear and abstract descriptions of how computations respecting call-time choice can proceed. They can be useful to explain basic operational aspects of functional logic languages to students or novice programmers, for instance. They have been used also as underlying formalisms to investigate other aspects of functional logic programming that need a clear notion of reduction; for instance, when proving essential properties of type systems, like subject reduction or progress. In addition, all the pieces are interconnected by strong theoretical results, which may be useful depending on the pursued goal.

Just like classical term rewriting and narrowing, the let-rewriting and narrowing relations define too broad computation spaces as to be adopted directly as concrete operational procedures of a programming language. To that purpose, they should be accompanied by a strategy that selects only certain computations. In this paper we have only given an example-driven discussion of strategies. We are quite confident that some known on-demand evaluation strategies, like lazy, needed or natural rewriting/narrowing, can be adapted to our formal setting. In (Riesco and Rodríguez-Hortalá 2010; Sánchez-Hernández 2011) we work out in more detail two concrete on-demand strategies for slight variants of let-rewriting and narrowing formalisms.

A subject of future work that might be of interest to the functional logic community is that of completing the comparison of different formalisms proposed in the field to capture call-time choice semantics: CRWL, admissible term graph rewriting/narrowing, natural semantics *à la* Launchbury, and let-rewriting/narrowing. Proving their equivalence would greatly enrich the set of tools available to the functional logic programming theoretician, since any known or future result obtained for one of the approaches could be applied to the rest on a sound technical basis.

References

Albert, E., Hanus, M., Huch, F., Oliver, J., and Vidal, G. 2005. Operational semantics for declarative multi-paradigm languages. *Journal of Symbolic Computation 40,* 1, 795–829.

Alpuente, M., Falaschi, M., Iranzo, P. J., and Vidal, G. 2003. Uniform lazy narrowing. *Journal of Logig and Computation 13,* 2, 287–312.

Antoy, S. 2005. Evaluation strategies for functional logic programming. *Journal of Symbolic Computation 40,* 1, 875–903.

Antoy, S., Brown, D., and Chiang, S. 2006. On the correctness of bubbling. In *17th International Conference on Rewriting Techniques and Applications (RTA'06).* Springer LNCS 4098, 35–49.

Antoy, S., Brown, D., and Chiang, S. 2007. Lazy context cloning for non-deterministic graph rewriting. *Electronic Notes in Theoretical Computer Science 176,* 1, 3–23.

Antoy, S., Echahed, R., and Hanus, M. 1994. A needed narrowing strategy. In *21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94).* ACM, 268–279.

Antoy, S., Echahed, R., and Hanus, M. 2000. A needed narrowing strategy. *Journal of the ACM 47,* 4, 776–822.

Antoy, S. and Hanus, M. 2000. Compiling multi-paradigm declarative programs into prolog. In *3rd International Workshop on Frontiers of Combining Systems (FroCoS'00).* Springer LNCS 1794, 171–185.

Antoy, S. and Hanus, M. 2006. Overlapping rules and logic variables in functional logic programs. In *22nd International Conference on Logic Programming (ICLP'06).* Springer LNCS 4079, 87–101.

Ariola, Z. M. and Arvind. 1995. Properties of a first-order functional language with sharing. *Theoretical Computer Science 146, 1&2*, 69–108.

Ariola, Z. M. and Felleisen, M. 1997. The call-by-need lambda calculus. *Journal of Functional Programming 7,* 3, 265–301.

Ariola, Z. M., Felleisen, M., Maraist, J., Odersky, M., and Wadler, P. 1995. The call-by-need lambda calculus. In *22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95).* ACM, 233–246.

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

Barendregt, H. P., Eekelen, M. C. J. D., Glauert, J. R. W., Kennaway, J. R., Plasmeijer, M. J., and Sleep, M. R. 1987. Term Graph Rewriting. In *1st Parallel Architectures and Languages Europe (PARLE'87), Volume II.* Springer LNCS 259, 141–158.

Brassel, B. and Huch, F. 2007. On a tighter integration of functional and logic programming. In *5th Asian Symposium on Programming Languages and Systems (APLAS'07).* Springer LNCS 4807, 122–138.

Caballero, R. and Sánchez, J., Eds. 2006. TOY: A multiparadigm declarative language, version 2.2.3. Technical report, Universidad Complutense de Madrid.

Cheong, P. and Fribourg, L. 1993. Implementation of narrowing: The Prolog-based approach. In *Logic programming languages: constraints, functions, and objects.* MIT Press, 1–20.

DeGroot, D. and Lindstrom, G. e. 1986. *Logic Programming, Functions, Relations, and Equations.* Prentice Hall.

Dios-Castro, J. and López-Fraguas, F. J. 2007. Extra variables can be eliminated from functional logic programs. *Electronic Notes in Theoretical Computer Science 188 188*, 3–19.

Echahed, R. and Janodet, J.-C. 1997. On constructor-based graph rewriting systems. Research Report 985-I, IMAG.

Echahed, R. and Janodet, J.-C. 1998. Admissible graph rewriting and narrowing. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*. MIT Press, 325 – 340.

Escobar, S., Meseguer, J., and Thati, P. 2005. Natural narrowing for general term rewriting systems. In *16th International Conference on Rewriting Techniques and Applications (RTA'05)*. Springer LNCS 3467, 279–293.

González-Moreno, J. C., Hortalá-González, T., López-Fraguas, F. J., and Rodríguez-Artalejo, M. 1996. A rewriting logic for declarative programming. In *6th European Symposium on Programming (ESOP'96)*. Springer LNCS 1058, 156–172.

González-Moreno, J. C., Hortalá-González, T., López-Fraguas, F. J., and Rodríguez-Artalejo, M. 1999. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming 40,* 1, 47–87.

González-Moreno, J. C., Hortalá-González, T., and Rodríguez-Artalejo, M. 1997. A higher order rewriting logic for functional logic programming. In *14th International Conference on Logic Programming (ICLP'97)*. MIT Press, 153–167.

Hanus, M. 1994. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming 19&20*, 583–628.

Hanus, M. 2007. Multi-paradigm declarative languages. In *23rd International Conference on Logic Programming (ICLP'07)*. Springer LNCS 4670, 45–75.

Hanus, M., Kuchen, H., and Moreno-Navarro, J. J. 1995. Curry: A truly functional logic language. In *Workshop on Visions for the Future of Logic Programming (ILPS'95)*. 95–107.

Hanus, M., Ed. 2006. Curry: An integrated functional logic language (version 0.8.2). Available at *http://www.informatik.uni-kiel.de/~curry/report.html.*

Hullot, J. 1980. Canonical forms and unification. In *5th Conference on Automated Deduction (CADE'80)*. Springer LNCS 87, 318–334.

Hussmann, H. 1993. *Non-Determinism in Algebraic Specifications and Algebraic Programs.* Birkhäuser Verlag.

Kutzner, A. and Schmidt-Schauss, M. 1998. A non-deterministic call-by-need lambda calculus. In *3th ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*. ACM SIGPLAN Notices 34(1), 324–335.

Launchbury, J. 1993. A natural semantics for lazy evaluation. In *20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'93)*. ACM, 144–154.

Loogen, R., López-Fraguas, F. J., and Rodríguez-Artalejo, M. 1993. A demand driven computation strategy for lazy narrowing. In *5th International Symposium on Programming Language Implementation and Logic Programming (PLILP'93)*. Springer LNCS 714, 184–200.

López-Fraguas, F. J., Martin-Martin, E., and Rodríguez-Hortalá, J. 2010a. Liberal typing for functional logic programs. In *8th Asian Symposium on Programming Languages and Systems (APLAS'10)*. Springer LNCS 6461, 80–96.

López-Fraguas, F. J., Martin-Martin, E., and Rodríguez-Hortalá, J. 2010b. New results on type systems for functional logic programming. In 18th International Workshop on Functional and (Constraint) Logic Programming (WFLP'09), Revised Selected Papers. Springer LNCS 5979, 128–144.

López-Fraguas, F. J. and Rodríguez-Hortalá, J. 2010. The full abstraction problem for higher order functional-logic programs. *CoRR, arXiv:1002:1833.*

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2007a. Equivalence of two formal semantics for functional logic programs. *Electronic Notes in Theoretical Computer Science 188 188*, 117–142.

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2007b. A simple rewrite notion for call-time choice semantics. In *9th International Conference on Principles and Practice of Declarative Programming (PPDP'07)*. ACM, 197–208.

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2008. Rewriting and call-time choice: the HO case. In *9th International Symposium on Functional and Logic Programming (FLOPS'08)*. Springer LNCS 4989, 147–162.

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2009a. A flexible framework for programming with non-deterministic functions. In *2009 ACM SIGPLAN Symposium on Partial Evaluation and Program Manipulation (PEPM'09)*. ACM, 91–100.

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2009b. A fully abstract semantics for constructor based term rewriting systems. In *20th International Conference on Rewriting Techniques and Applications (RTA'09)*. Springer LNCS 5595, 320–334.

López-Fraguas, F. J., Rodríguez-Hortalá, J., and Sánchez-Hernández, J. 2009c. Narrowing for First Order Functional Logic Programs with Call-Time Choice Semantics. In *17th International Conference on Applications of Declarative Programming and Knowledge Management (INAP'07) and 21st Workshop on (Constraint) Logic Programming (WLP'07), Revised Selected Papers*. Springer LNAI 5437, 206–222.

López-Fraguas, F. J. and Sánchez-Hernández, J. 1999. $\mathcal{TOY}$: A multiparadigm declarative system. In *10th International Conference on Rewriting Techniques and Applications (RTA'99)*. Springer LNCS 1631, 244–247.

López-Fraguas, F. J. and Sánchez-Hernández, J. 2001. Functional logic programming with failure: A set-oriented view. In *8th International Conference on Logic for Programming and Automated Reasoning (LPAR'01)*. Springer LNAI 2250, 455–469.

Maraist, J., Odersky, M., and Wadler, P. 1998. The call-by-need lambda calculus. *Journal of Functional Programming 8,* 3, 275–317.

McCarthy, J. 1963. A Basis for a Mathematical Theory of Computation. In *Computer Programming and Formal Systems*. North-Holland, 33–70.

Moreno-Navarro, J. J. and Rodríguez-Artalejo, M. 1992. Logic programming with functions and predicates: The language Babel. *Journal of Logic Programming 12*, 189–223.

Plump, D. 1998. Term graph rewriting. Report CSI-R9822, Computing Science Institute, University of Nijmegen.

Plump, D. 2001. Essentials of term graph rewriting. *Electronic Notes Theoretical Computer Science 51*, 277–289.

Riesco, A. and Rodríguez-Hortalá, J. 2010. Programming with singular and plural non-deterministic functions. In *2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM'10)*. ACM, 83–92.

Rodríguez-Artalejo, M. 2001. Functional and constraint logic programming. In *Revised Lectures of the International Summer School CCL'99*. Springer LNCS 2002, 202–270.

Sánchez-Hernández, J. 2004. Una aproximación al fallo constructivo en programación declarativa multiparadigma. Ph.D. thesis, Departamento Sistemas Informáticos y Programación, Universidad Complutense de Madrid.

Sánchez-Hernández, J. 2011. Reduction strategies for rewriting with call-time choice. In *11th Jornadas sobre Programación y Lenguajes (PROLE'11)*.

Schmidt-Schauss, M. and Machkasova, E. 2008. A finite simulation method in a nondeterministic call-by-need lambda-calculus with letrec, constructors, and case. In *19th International Conference on Rewriting Techniques and Applications (RTA'08)*. Springer LNCS 5117, 321–335.

Sondergaard, H. and Sestoft, P. 1990. Referential transparency, definiteness and unfoldability. *Acta Informatica 27,* 6, 505–517.

Søndergaard, H. and Sestoft, P. 1992. Non-determinism in functional languages. *The Computer Journal 35,* 5, 514–523.

Vado-Vírseda, R. d. 2002. Estrategias de estrechamiento perezoso. Trabajo de Investigación de Tercer Ciclo, Dpto. de Sistemas Informáticos y Programación, Universidad Complutense de Madrid.

Vado-Vírseda, R. d. 2003. A demand-driven narrowing calculus with overlapping definitional trees. In *5th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'03)*. ACM, 213–227.

Zartmann, F. 1997. Denotational abstract interpretation of functional logic programs. In *4th International Symposium on Static Analysis (SAS'97)*. Springer LNCS 1302, 141–159.

Appendix A Detailed proofs for the results

In the proofs we will use the usual notation for positions, subexpressions and repacements from (Baader and Nipkow 1998). The *set of positions* of an expression $e \in Exp$ is a set $O(e)$ of strings of positive integers defined as:

- If $e \equiv X \in \mathcal{V}$, then $O(e) = \epsilon$, where ϵ is the empty string.
- If $e \equiv h(e_1, \ldots, e_n)$ with $h \in \Sigma$, then

$$O(e) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in O(e_i)\}$$

The *subexpression of* e *at position* $p \in O(e)$, denoted $e|_p$, is defined as:

$$\begin{array}{rcl} e|_\epsilon & = & e \\ h(e_1, \ldots, e_n)|_{ip} & = & e_i|_p \end{array}$$

For a position $p \in O(e)$, we define the *replacement of the subexpression of* e *at position* p *by* e' —denoted $e[e']_p$— as follows:

$$\begin{array}{rcl} e[e']_\epsilon & = & e' \\ h(e_1, \ldots, e_n)[e']_{ip} & = & h(e_1, \ldots, e_i[e']_p, \ldots, e_n) \end{array}$$

When performing proofs by induction we will usually use IH to refer to the induction hypothesis of the current induction. We will use an asterisk to denote the use of a let-rewriting rule one or more times, as in (Flat*). We will also use the following auxiliary results.

A.1 Lemmas

The following lemmas are used in the proofs for the results in the article. Most of them are straightforwardly proved by induction, so we only detail the proof in the interesting cases.

Lemma 17

$\forall t \in CTerm_\perp.\ |t| = t.$

Lemma 18

$\forall t \in CTerm_\perp.\ \mathcal{P} \vdash_{CRWL_{let}} t \rightarrow t.$

Lemma 19

Given $\theta, \theta' \in LSubst_\perp$, $e \in LExp_\perp$, if $\theta \sqsubseteq \theta'$ then $e\theta \sqsubseteq e\theta'$.

Lemma 20

Given $\theta \in LSubst_\perp$, $e, e' \in LExp_\perp$, if $e \sqsubseteq e'$ then $e\theta \sqsubseteq e'\theta$.

Lemma 21

For every $e, e' \in LExp_\perp$, $\mathcal{C} \in Cntxt$, if $|e| \sqsubseteq |e'|$ then $|\mathcal{C}[e]| \sqsubseteq |\mathcal{C}[e']|$.

Proof

We proceed by induction on the structure of $\mathcal{C}$. The base case is straightforward because of the hypothesis. For the Inductive Step we have:

- $\mathcal{C} \equiv h(\ldots, \mathcal{C}', \ldots)$. Directly by IH.
- $\mathcal{C} \equiv let\ X = \mathcal{C}'\ in\ e_1$, so $\mathcal{C}[e] \equiv let\ X = \mathcal{C}'[e]\ in\ e_1$. Then:

$$\begin{array}{l} |\mathcal{C}[e]| = |let\ X = \mathcal{C}'[e]\ in\ e_1| = |e_1|[X/|\mathcal{C}'[e]|] \\ \sqsubseteq_{IH^{(*)}} |e_1|[X/|\mathcal{C}'[e']|] = |let\ X = \mathcal{C}'[e']\ in\ e_1| = |\mathcal{C}[e']| \end{array}$$

  $(*)$ By IH we have $|\mathcal{C}'[e]| \sqsubseteq |\mathcal{C}'[e']|$, therefore $[X/|\mathcal{C}'[e]|] \sqsubseteq [X/|\mathcal{C}'[e']|]$. Finally, by Lemma 19, $|e_1|[X/|\mathcal{C}'[e]|] \sqsubseteq |e_1|[X/|\mathcal{C}'[e']|]$.
- $\mathcal{C} \equiv let\ X = e_1\ in\ \mathcal{C}'$. Similar to the previous case but using Lemma 20 to obtain $|\mathcal{C}'[e]|\ [X/|e_1|] \sqsubseteq |\mathcal{C}'[e']|[X/|e_1|]$ from the IH $|\mathcal{C}'[e]| \sqsubseteq |\mathcal{C}'[e']|$.

□

Lemma 22

If $|e| = |e'|$ then $|\mathcal{C}[e]| = |\mathcal{C}[e']|$

Proof

Since $\sqsubseteq$ is a partial order, we know by reflexivity that $|e| \sqsubseteq |e'|$ and $|e'| \sqsubseteq |e|$. Then by Lemma 21 we have $|\mathcal{C}[e]| \sqsubseteq |\mathcal{C}[e']|$ and $|\mathcal{C}[e']| \sqsubseteq |\mathcal{C}[e]|$. Finally, by antisymmetry of the partial order $\sqsubseteq$ we have that $|\mathcal{C}[e]| = |\mathcal{C}[e']|$. □

Lemma 23

For all $e_1, e_2 \in LExp, X \in \mathcal{V}$, $|e_1[X/e_2]| \equiv |e_1|[X/|e_2|]$

Proof
By induction on the structure of e_1. The most interesting case is when $e_1 \equiv let\ Y = s_1\ in\ s_2$. By the variable convention $Y \notin dom([X/e_2])$ and $Y \notin vran([X/e_2])$, so:

$$\begin{array}{ll} |e_1[X/e_2]| \equiv |let\ Y = s_1[X/e_2]\ in\ s_2[X/e_2]| & \\ \equiv |s_2[X/e_2]|[Y/|s_1[X/e_2]|] & \\ \equiv_{IH} |s_2|[X/|e_2|][Y/(|s_1|[X/|e_2|])] & \\ \equiv |s_2|[Y/|s_1|][X/|e_2|] & (*) \\ \equiv |let\ Y = s_1\ in\ s_2|[X/|e_2|] \equiv |e_1|[X/|e_2|] & \end{array}$$

(*) Using Lemma 1 with the matching $[e/|s_2|, \theta/[X/|e_2|], X/Y, e'/|s_1|]$. □

Lemma 24
Given $\theta \in LSubst_\perp$, $e, e' \in LExp_\perp$, if $e \sqsubseteq e'$ then $e\theta \sqsubseteq e'\theta$.

Lemma 25
For every $\sigma \in LSubst_\perp, \mathcal{C} \in Cntxt$ and $e \in LExp_\perp$ such that $(dom(\sigma) \cup vran(\sigma)) \cap BV(\mathcal{C}) = \emptyset$ we have that $(\mathcal{C}[e])\sigma \equiv \mathcal{C}\sigma[e\sigma]$.

Proof
By induction on the structure of $\mathcal{C}$. The most interesting cases are those concerning let-expressions:

- $\mathcal{C} \equiv let\ X = \mathcal{C}'\ in\ e_1$: therefore $\mathcal{C}[e] \equiv let\ X = \mathcal{C}'[e]\ in\ e_1$. Then

$$(\mathcal{C}[e])\sigma \equiv let\ X = (\mathcal{C}'[e])\sigma\ in\ e_1\sigma \equiv_{IH}^{(*)} let\ X = \mathcal{C}'\sigma[e\sigma]\ in\ e_1\sigma$$
$$\equiv (let\ X = (\mathcal{C}'[])\sigma\ in\ e_1\sigma)[e\sigma] \equiv^{(**)} ((let\ X = \mathcal{C}'[]\ in\ e_1)\sigma)[e\sigma] \equiv \mathcal{C}\sigma[e\sigma]$$

(∗): by definition $BV(let\ X = \mathcal{C}'\ in\ e) = BV(\mathcal{C}')$, so $(dom(\sigma) \cup vran(\sigma)) \cap BV(\mathcal{C}) = \emptyset = (dom(\sigma) \cup vran(\sigma)) \cap BV(\mathcal{C}')$.
(∗∗): we can apply the last step because by hypothesis we can assure that we do not need any renaming to apply $(let\ X = \mathcal{C}'[]\ in\ e_1)\sigma$.

- $\mathcal{C} \equiv let\ X = e_1\ in\ \mathcal{C}'$: therefore $\mathcal{C}[e] \equiv let\ X = e_1\ in\ \mathcal{C}'[e]$. Then

$$(\mathcal{C}[e])\sigma \equiv let\ X = e_1\sigma\ in\ (\mathcal{C}'[e])\sigma \equiv_{IH} let\ X = e_1\sigma\ in\ \mathcal{C}'\sigma[e\sigma]$$
$$\equiv (let\ X = e_1\sigma\ in\ (\mathcal{C}'[])\sigma)[e\sigma] \equiv^{(*)} ((let\ X = e_1\ in\ \mathcal{C}'[])\sigma)[e\sigma] \equiv \mathcal{C}\sigma[e\sigma]$$

(∗): we can apply the last step because by hypothesis we can assure that we do not need any renaming to apply $(let\ X = e_1\ in\ \mathcal{C}'[])\sigma$.

□

Lemma 26
For any $e \in Exp_\perp$, $t \in CTerm_\perp$ and program $\mathcal{P}$, if $\mathcal{P} \vdash e \rightarrowtriangle t$ then there is a derivation for $\mathcal{P} \vdash e \rightarrowtriangle t$ in which every free variable used belongs to $FV(e \rightarrowtriangle t)$.

Proof
A simple extension of the proof in (Dios-Castro and López-Fraguas 2007). □

Lemma 27

For every $CRWL_{let}$ derivation $e \rightarrowtriangle t$ there exists $e' \in LExp_\perp$ which is syntactically equivalent to e module α-conversion, and a $CRWL_{let}$ derivation for $e' \rightarrowtriangle t$ such that if $\mathcal{B}$ is the set of bound variables used in $e' \rightarrowtriangle t$ and $\mathcal{E}$ is the set of free variables used in the instantiation of extra variables in $e' \rightarrowtriangle t$ then $\mathcal{B} \cap (\mathcal{E} \cup var(t)) = \emptyset$.

Proof

By Lemma 26, if $\mathcal{F}$ is the set of free variables used in $e' \rightarrowtriangle t$, then $\mathcal{F} \subseteq FV(e' \rightarrowtriangle t)$, in fact $\mathcal{F} = FV(e' \rightarrowtriangle t)$, as $FV(e')$ and $FV(t)$ are used in the top derivation of the derivation tree for $e' \rightarrowtriangle t$. As by definition $\mathcal{E} \cup var(t) \subseteq \mathcal{F}$, if we prove $\mathcal{B} \cap \mathcal{F} = \emptyset$ then $\mathcal{B} \cap (\mathcal{E} \cup var(t)) = \emptyset$ is a trivial consequence. To prove that we will prove that for every $a \in LExp_\perp$ used in the derivation for $e' \rightarrowtriangle t$ we have $BV(a) \cap FV(a) = \emptyset$. We can build e' using α-conversion to ensure that $BV(e') \cap FV(e') = \emptyset$. This can be easily maintained as an invariant during the derivation, as the new let-bindings that appear during the derivation are those introduced in the instances of the rule used during the **OR** steps, and be can ensure by α-conversion that $BV(a) \cap FV(a) = \emptyset$ for these instances too, as α-conversion leaves the hypersemantics untouched. $\square$

A.2 Proofs for Section 2.2

Theorem 1 (Compositionality of CRWL)

For any $\mathcal{C} \in Cntxt$, $e, e' \in Exp_\perp$

$$[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!]$$

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt. [\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$

Proof

We prove that $\mathcal{C}[e] \rightarrowtriangle t \Leftrightarrow \exists s \in CTerm_\perp$ such that $e \rightarrowtriangle s$ and $\mathcal{C}[s] \rightarrowtriangle t$.

$\Rightarrow$) Induction on the size of the proof for $\mathcal{C}[e] \rightarrowtriangle t$.

Base case The base case only allows the proofs $\mathcal{C}[e] \rightarrowtriangle \perp$ using (B), $\mathcal{C}[e] \equiv X \rightarrowtriangle X$ using (RR) and $\mathcal{C}[e] \equiv c \rightarrowtriangle c$ with $c \in CS$ using (DC), that are clear. When $\mathcal{C} = [\,]$ the proof is trivial with $s = t$ and using Lemma 18.

Inductive step Direct application of the IH.

$\Leftarrow$) By induction on the size of the proof for $\mathcal{C}[s] \rightarrowtriangle t$

Base case The base case only allows the proofs $\mathcal{C}[s] \rightarrowtriangle \perp$, $\mathcal{C}[s] \equiv X \rightarrowtriangle X$ and $\mathcal{C}[s] \equiv c \rightarrowtriangle c$ with $c \in CS$, that are clear. When $\mathcal{C} = [\,]$ we have that $\exists s \in CTerm_\perp$ such that $e \rightarrowtriangle s$ and $s \rightarrowtriangle t$. Since $s \rightarrowtriangle t$ by Lemma 5 we have $t \sqsubseteq s$, and using Proposition 3 $e \rightarrowtriangle t$ —as $e \sqsubseteq e$ because $\sqsubseteq$ is a partial order.

Inductive step Direct application of the IH.

$\square$

A.3 Proofs for Section 3

Theorem 3
Let $\mathcal{P}$ be a CRWL-program, $e \in Exp_\perp$ and $t \in CTerm_\perp$. Then:

$$\mathcal{P} \vdash_{CRWL} e \mathrel{-\!\!\triangleright} t \textit{ iff } e \rightarrowtail^*_{\mathcal{P}} t$$

Proof
It is easy to see that $\rightarrowtail^*$ coincides with the relation defined by the *BRC*-proof calculus of (González-Moreno et al. 1999), that is, $\mathcal{P} \vdash_{BRC} e \mathrel{-\!\!\triangleright} e' \;\leftrightarrow e \rightarrowtail^* e'$. But in that paper it is proved that *BRC*-derivability and CRWL-derivability (called there *GORC*-derivability) are equivalent. □

A.4 Proofs for Section 4

Lemma 1 (Substitution lemma for let-expressions)
Let $e, e' \in LExp_\perp$, $\theta \in Subst_\perp$ and $X \in \mathcal{V}$ such that $X \notin dom(\theta) \cup vran(\theta)$. Then:

$$(e[X/e'])\theta \equiv e\theta[X/e'\theta]$$

Proof
By induction over the structure of e. The most interesting cases are the base cases:

- $e \equiv X$: Then $(e[X/e'])\theta \equiv (X[X/e'])\theta \equiv e'\theta \equiv X[X/e'\theta]$
  $\equiv_{X \notin dom(\theta)} X\theta[X/e'\theta] \equiv e\theta[X/e'\theta]$
- $e \equiv Y \not\equiv X$: Then $(e[X/e't])\theta \equiv (Y[X/e'])\theta \equiv Y\theta$
  $\equiv_{X \notin ran(\theta)} Y\theta[X/e'\theta] \equiv e\theta[X/e'\theta]$

□

A.5 Proofs for Section 4.1

Lemma 2 (Closedness under CSubst of let-rewriting)
For any $e, e' \in LExp$, $\theta \in CSubst$ we have that $e \to^{l\,n} e'$ implies $e\theta \to^{l\,n} e'\theta$.

Proof
We prove that $e \to^l e'$ implies $e\theta \to^l e'\theta$ by a case distinction over the rule of the let-rewriting calculus applied:

(Fapp) Assume $f(t_1, \ldots, t_n) \to^l r$, using $(f(p_1, \ldots, p_n) \to e) \in \mathcal{P}$ and $\sigma \in CSubst$ such that $\forall i.p_i\sigma = t_i$ and $e\sigma = r$. But since $\sigma\theta \in CSubst$ and $\forall i.p_i\sigma\theta = t_i\theta$ then we can perform a (Fapp) step $f(t_1, \ldots, t_n)\theta \equiv f(t_1\theta, \ldots, t_n\theta) \to^l e\sigma\theta \equiv r\theta$.

(LetIn) Easily since $X \notin dom(\theta)$ because X is fresh.

(Bind) Assume $let\ X = t\ in\ e \to^l e[X/t]$ and some $\theta \in CSubst$. Then $t \in CTerm$ by the conditions of (Bind), hence $t\theta \in CTerm$ too and we can perform a (Bind) step $(let\ X = t\ in\ e)\theta \equiv let\ X = t\theta\ in\ e\theta \to^l e\theta[X/t\theta]$. Besides $X \notin (dom(\theta) \cup vran(\theta))$ by the variable convention, and so $e\theta[X/t\theta] \equiv e[X/t]\theta$ by Lemma 1, so are done.

(Elim) Easily as $X \notin FV(e_2\theta)$ because $X \notin vran(\theta)$ by the variable convention.

(Flat) Similar to the previous case since $Y \notin FV(e_3\theta)$.

(Contx) Assume $\mathcal{C}[e] \to^l \mathcal{C}[e']$ because $e \to^{l\,\prime} e'$ by one of the previous rules, and some $\theta \in CSubst$. Then we have already proved that $e\theta \to^l e'\theta$. Besides by the variable convention we have $BV(\mathcal{C}) \cap (dom(\theta) \cup vran(\theta)) = \emptyset$, hence by Lemma 25 $(\mathcal{C}[e])\theta \equiv \mathcal{C}\theta[e\theta]$. Furthermore, if $e \to^l e'$ was a (Fapp) step using $\sigma \in CSubst$ to build the instance of the program rule $(f(\overline{p})\sigma \to r\sigma)$, then $vran(\sigma|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$ by the conditions of (Contx), and therefore $vran((\sigma\theta)|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$. But as $\sigma\theta$ is the substitution used in the (Fapp) step $e\theta \to^l e'\theta$, then $\mathcal{C}\theta[e\theta] \to^l \mathcal{C}\theta[e'\theta]$ by (Contx). On the other hand, if $e \to^l e'$ was not a (Fapp) step then $\mathcal{C}\theta[e\theta] \to^l \mathcal{C}\theta[e'\theta]$ too, and finally we can apply Lemma 25 again to get $\mathcal{C}\theta[e'\theta] \equiv (\mathcal{C}[e'])\theta$.

The proof for $e \to^{l\,n} e'$ proceeds straightforwardly by induction on the length n of the derivation. □

Proposition 2 (Termination of $\to^{lnf}$)
Under any program we have that $\to^{lnf}$ is terminating.

Proof
We define for any $e \in LExp$ the size (k_1, k_2, k_3), where

$k_1 \equiv$ *number of subexpressions in e to which (LetIn) is applicable.*
$k_2 \equiv$ *number of* lets *in e.*
$k_3 \equiv$ *sum of the levels of nesting of all let-subexpressions in e.*

Sizes are lexicographically ordered. We prove now that application of *(LetIn), (Bind), (Elim), (Flat)* in any context (hence, also the application of (Contxt)) decreases the size, what proves termination of $\to^{lnf}$. The effect of each rule in the size is summarized as follows (in each case, we stop at the decreasing component):

$$\begin{array}{ll} \textit{(LetIn):} & (<, \_, \_) \\ \textit{(Bind):} & (=, <, \_) \\ \textit{(Elim):} & (\leq, <, \_) \\ \textit{(Flat):} & (=, =, <) \end{array}$$

□

Lemma 3 (Peeling lemma)
For any $e, e' \in LExp$ if $e \downarrow^{lnf} e'$ —i.e, e' is a $\to^{lnf}$ normal form for e— then e' has the shape $e' \equiv let\ \overline{X = f(\overline{t})}\ in\ e''$ such that $e'' \in \mathcal{V}$ or $e'' \equiv h(\overline{t'})$ with $h \in \Sigma$, $\overline{f} \subseteq FS$ and $\overline{t}, \overline{t'} \subseteq CTerm$.
Moreover if $e \equiv h(e_1, \ldots, e_n)$ with $h \in \Sigma$, then

$$e \equiv h(e_1, \ldots, e_n) \to^{lnf^*} let\ \overline{X = f(\overline{t})}\ in\ h(t_1, \ldots, t_n) \equiv e'$$

under the conditions above, and verifying also that $t_i \equiv e_i$ whenever $e_i \in CTerm$.

Proof
We prove it by contraposition: if an expression e does not have that shape, e is not a $\to^{lnf}$ normal form. We define the set of expressions which are not cterms as:

$$\begin{array}{rl} nt ::= & c(\ldots, nt, \ldots) \\ & \mid f(\overline{e}) \\ & \mid let\ X = e_1\ in\ e_2 \end{array}$$

We also define the set of expressions which do not have the presented shape recursively as:

$$\begin{array}{rl} ne ::= & h(\ldots, nt, \ldots) \\ & \mid let\ X = f(\overline{t})\ in\ ne \\ & \mid let\ X = f(\ldots, nt, \ldots)\ in\ e \\ & \mid let\ X = c(\overline{e})\ in\ e \\ & \mid let\ X = (let\ Y = e'\ in\ e'')\ in\ e \end{array}$$

We prove by induction on the structure of an expression ne that it is always possible to perform a $\rightarrow^{lnf}$ step:

Base case:

- $ne \equiv h(\ldots, nt, \ldots)$: there are various cases depending on nt:
  - at some depth the non-cterm will contain a subexpression $c'(\ldots, nt', \ldots)$ where nt' is a function application $f(\overline{e})$ or a let-rooted expression $let\ X = e_1\ in\ e_2$. Therefore we can apply the rule (Contx) with (LetIn) in that position.
  - $f(\overline{e})$: we can apply the rule (LetIn) and perform the step

  $$h(\ldots, f(\overline{e}), \ldots) \rightarrow^{lnf} let\ X = f(\overline{e})\ in\ h(\ldots, X, \ldots)$$

  - $let\ X = e_1\ in\ e_2$: the same as the previous case.
- $let\ X = f(\ldots, nt, \ldots)\ in\ e$: we can perform a (Contx) with (LetIn) step in $f(\ldots, nt, \ldots)$ as in the previous $h(\ldots, nt, \ldots)$ case.
- $let\ X = c(\overline{e})\ in\ e$: if $\overline{e}$ are cterms $\overline{t}$, then $c(\overline{t})$ is a cterm and we can perform a (Bind) step $let\ X = c(\overline{t})\ in\ e \rightarrow^{lnf} e[X/c(\overline{t})]$. If $\overline{e}$ contains any expression ne then we can perform a (Contx) with (LetIn) step as in the previous $h(\ldots, nt, \ldots)$ case.
- $let\ X = (let\ Y = e'\ in\ e'')\ in\ e$: by the variable convention we can assume that $Y \notin FV(e)$, so we can perform a (Flat) step $let\ X = (let\ Y = e'\ in\ e'')\ in\ e \rightarrow^{lnf} let\ Y = e'\ in\ let\ X = e''\ in\ e$.

Inductive step:

- $let\ X = f(\overline{t})\ in\ ne$: by IH we have that $ne \rightarrow^{lnf} ne'$, so by the rule (Contx) we can perform a step $let\ X = f(\overline{t})\ in\ ne \rightarrow^{lnf} let\ X = f(\overline{t})\ in\ ne'$.

Notice that if the original expression has the shape $h(e_1, \ldots, e_n)$ the arguments e_i which are cterms remain unchanged in the same position. The reason is that no rule can affect them: the only rule applicable at the top is (LetIn), and it can not place them in a let binding outside $h(\ldots)$; besides cterms do not match with the left-hand side of any rule, so they can not be rewritten by any rule. □

Lemma 4 (Growing of shells)
Under any program $\mathcal{P}$ and for any $e, e' \in LExp$

i) $e \rightarrow^{l^*} e'$ implies $|e| \sqsubseteq |e'|$
ii) $e \rightarrow^{lnf^*} e'$ implies $|e| \equiv |e'|$

Proof for Lemma 4
We prove the lemma for one step ($e \rightarrow^l e'$ and $e \rightarrow^{lnf} e'$) by a case distinction over the rule of the let-rewriting calculus applied:

(Fapp) The step is $f(t_1, \dots, t_n) \rightarrow^l r$, and $|f(t_1, \dots, t_n)| = \perp \sqsubseteq |r|$.

(LetIn) The equality $|h(e_1, \dots, e, \dots, e_n)| = |let\ X = e\ in\ h(e_1, \dots, X, \dots, e_n)|$ follows easily by a case distinction on h.

(Bind) The step is $let\ X = t\ in\ e \rightarrow^l e[X/t]$, so $|let\ X = t\ in\ e| = |e|[X/|t|] = |e[X/t]|$ by Lemma 23.

(Elim) The step is $let\ X = e_1\ in\ e_2 \rightarrow^l e_2$ with $X \notin FV(e_2)$. Then $|let\ X = e_1\ in\ e_2| = |e_2|[X/|e_1|] = |e_2|$. Since the variables in the shell of an expression is a subset of the variables in the original expression, we can conclude that if $X \notin FV(e_2)$ then $X \notin FV(|e_2|)$.

(Flat) The step is $let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \rightarrow^l let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3)$ with $Y \notin FV(e_3)$. By the variable convention we can assume that $X \notin FV(let\ Y = e_1\ in\ e_2)$ —in particular $X \notin FV(e_1)$. Then:

$$\begin{array}{l} |let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3)| \\ = |let\ X = e_2\ in\ e_3|[Y/|e_1|] \\ = (|e_3|[X/|e_2|])[Y/|e_1|] \end{array}$$

Notice that $X \notin dom([Y/|e_1|])$ and $X \notin vran([Y/|e_1|]) = FV(|e_1|)$ because $X \notin FV(e_1)$ and $FV(|e_1|) \subseteq FV(e_1)$. Therefore we can use Lemma 1:

$$\begin{array}{ll} (|e_3|[X/|e_2|])[Y/|e_1|] & \\ = (|e_3|[Y/|e_1|])[X/(|e_2|[Y/|e_1|])] & \text{By Lemma 1} \\ = |e_3|[X/(|e_2|[Y/|e_1|])] & Y \notin FV(e_3), \text{ so } Y \notin FV(|e_3|) \\ = |e_3|[X/|let\ Y = e_1\ in\ e_2|] & \\ = |let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3| & \end{array}$$

(Contx) The step is $\mathcal{C}[e] \rightarrow^l \mathcal{C}[e']$ with $e \rightarrow^l e'$ using any of the previous rules. Then we have $|e| \sqsubseteq |e'|$, and by Lemma 21 $\mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$. If the step is $\mathcal{C}[e] \rightarrow^{lnf} \mathcal{C}[e']$ then rule (Fapp) has not been used in the reduction $e \rightarrow^{lnf} e'$ and by the previous rules we have $|e| = |e'|$. In that case by Lemma 22 we have $\mathcal{C}[e] = \mathcal{C}[e']$.

The extension of this result to $\rightarrow^{l^*}$ and $\rightarrow^{lnf^*}$ is a trivial induction over the number of steps of the derivation. □

A.6 Proofs for Section 4.2

Theorem 4 (CRWL vs. CRWL$_{let}$)
For any program $\mathcal{P}$ without lets, and any $e \in Exp_\perp$:

$$[\![e]\!]^{\mathcal{P}}_{CRWL} = [\![e]\!]^{\mathcal{P}}_{CRWL_{let}}$$

Proof
As any calculus rule from CRWL is also a rule from CRWL$_{let}$, then any CRWL-proof is also a CRWL$_{let}$-proof, therefore $[\![e]\!]_{CRWL} \subseteq [\![e]\!]_{CRWL_{let}}$. For the other inclusion, assume no let-binding is present in the program and let $e \in Exp$. Then, for any

$t \in CTerm_\perp$, as the rules of CRWL$_{let}$ do not introduce any let-binding and the rule (Let) is only used for let-rooted expressions, the CRWL$_{let}$-proof $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$ will be also a CRWL-proof for $\mathcal{P} \vdash_{CRWL_{let}} e \rightarrowtriangle t$, hence $[\![e]\!]_{CRWL_{let}} \subseteq [\![e]\!]_{CRWL}$ too. $\square$

The following Lemma is used to prove point *iii)* of Lemma 5. Notice that this Lemma uses the notions of hyperdenotation ($[\![\![\ ]\!]\!]$) and hyperinclusion ($\Subset$) presented in the final part of Section 4.2.

Lemma 28
Under any program $\mathcal{P}$ and for any $e \in LExp_\perp$ we have that $[\![\![e]\!]\!] \Subset \lambda\theta.(|e\theta|\!\uparrow)\!\downarrow$.

Proof
We will use the following equivalent characterization of $(e\!\uparrow)\!\downarrow$:

$$(e\!\uparrow)\!\downarrow = \{e_1 \in LExp_\perp \mid \exists e_2 \in LExp_\perp.\ e \sqsubseteq e_2 \wedge e_1 \sqsubseteq e_2\}$$

note that $\{e_2 \in LExp_\perp \mid e \sqsubseteq e_2\}$ is precisely the set $e\!\uparrow$. Besides note that:

$$\begin{array}{l} [\![\![e]\!]\!] \Subset \lambda\theta.(|e\theta|\!\uparrow)\!\downarrow \\ \Leftrightarrow \forall\theta \in CSubst_\perp.\ [\![e\theta]\!] \subseteq (|e\theta|\!\uparrow)\!\downarrow \\ \Leftrightarrow \forall\theta \in CSubst_\perp, t \in CTerm_\perp.\ e\theta \rightarrowtriangle t \\ \qquad \Rightarrow t \in (|e\theta|\!\uparrow)\!\downarrow \\ \Leftrightarrow \forall\theta \in CSubst_\perp, t \in CTerm_\perp.\ e\theta \rightarrowtriangle t \\ \qquad \Rightarrow \exists t' \in CTerm_\perp.\ |e\theta| \sqsubseteq t' \wedge t \sqsubseteq t' \end{array}$$

where $t' \in CTerm_\perp$ is implied by $|e\theta| \sqsubseteq t'$. To prove this last formulation first consider the case when $t \equiv \perp$. Then we are done with $t' \equiv |e\theta|$ because then $|e\theta| \sqsubseteq |e\theta| \equiv t'$ and $t \equiv \perp \sqsubseteq |e\theta| \equiv t'$.

For the other case we proceed by induction on the structure of e. Regarding the base cases:

- If $e \equiv \perp$ then $t \equiv \perp$ and we are in the previous case.
- If $e \equiv X \in \mathcal{V}$ then $e\theta \equiv \theta(X) \rightarrowtriangle t$, and as $\theta \in CSubst_\perp$ then $\theta(X) \in CTerm_\perp$ which implies $t \sqsubseteq \theta(X)$ by Lemma 5. But then we can take $t' \equiv \theta(X)$ for which $t \sqsubseteq \theta(X) \equiv t'$ and $|e\theta| \equiv |\theta(X)| \equiv \theta(X)$ —by Lemma 17 since $\theta(X) \in CTerm_\perp$—, and $\theta(X) \sqsubseteq \theta(X) \equiv t'$.
- If $e \equiv c \in DC$ then either $t \equiv \perp$ and we are in the previous case, or $t \equiv c$. But then we can take $t' \equiv c$ for which $|e\theta| \equiv c \sqsubseteq c \equiv t'$, and $t \equiv c \sqsubseteq c \equiv t'$.
- If $e \equiv f \in FS$ then $|e\theta| \equiv |f| \equiv \perp$, and so $|e\theta|\!\uparrow = CTerm_\perp$ and $(|e\theta|\!\uparrow)\!\downarrow = CTerm_\perp \supseteq [\![e\theta]\!]$, so we are done.

Concerning the inductive steps:

- If $e \equiv f(e_1, \ldots, e_n)$ for $f \in FS$ then $|e\theta| \equiv \perp$ and we proceed like in the case for $e \equiv f$.
- If $e \equiv c(e_1, \ldots, e_n)$ for $c \in DC$ then either $t \equiv \perp$ and we are in the previous case, or $t \equiv c(t_1, \ldots, t_n)$ such that $\forall i.\ e_i\theta \rightarrowtriangle t_i$. But then by IH we get $\forall i.\ \exists t'_i.\ |e_i\theta| \sqsubseteq t'_i \wedge t_i \sqsubseteq t'_i$, so we can take $t' \equiv c(t'_1, \ldots, t'_n)$ for which $|e\theta| \equiv c(|e_1\theta|, \ldots, |e_n\theta|) \sqsubseteq c(t'_1, \ldots, t'_n) \equiv t'$ and $t \equiv c(t_1, \ldots, t_n) \sqsubseteq c(t'_1, \ldots, t'_n) \equiv t'$.

- If $e \equiv let\ X = e_1\ in\ e_2$ then either $t \equiv \perp$ and we are in the previous case, or we have the following proof:

$$\frac{e_1\theta \rightarrowtriangle t_1 \quad e_2\theta[X/t_1] \rightarrowtriangle t}{e\theta \equiv let\ X = e_1\theta\ in\ e_2\theta \rightarrowtriangle t}\ Let$$

Then by IH over e_1 we get that $\exists t'_1.\ |e_1\theta| \sqsubseteq t'_1 \wedge t_1 \sqsubseteq t'_1$. Hence $[X/t_1] \sqsubseteq [X/t'_1]$ so by Proposition 5 we have that $e_2\theta[X/t_1] \rightarrowtriangle t$ implies $e_2\theta[X/t'_1] \rightarrowtriangle t$. But then we can apply the IH over e_2 with $\theta[X/t'_1]$ to get some $t' \in CTerm_\perp$ such that $t \sqsubseteq t'$ and $|e_2\theta[X/t'_1]| \sqsubseteq t'$, which implies:

$$\begin{array}{ll}
t' \sqsupseteq |e_2\theta[X/t'_1]| & \\
\equiv |e_2\theta|[X/|t'_1|] & \text{by Lemma 23} \\
\equiv |e_2\theta|[X/t'_1] & \text{by Lemma 17 as } t'_1 \in CTerm_\perp \\
\sqsupseteq |e_2\theta|[X/|e_1\theta|] & \text{as } |e_1\theta| \sqsubseteq t'_1 \\
\equiv |let\ X = e_1\theta\ in\ e_2\theta| \equiv |e\theta| &
\end{array}$$

□

Lemma 5
For any program $e \in LExp_\perp$, $t, t' \in CTerm_\perp$:

1. $t \rightarrowtriangle t'$ iff $t' \sqsubseteq t$.
2. $|e| \in [\![e]\!]$.
3. $[\![e]\!] \subseteq (|e|{\uparrow}){\downarrow}$, where for a given $E \subseteq LExp_\perp$ its upward closure is $E{\uparrow} = \{e' \in LExp_\perp |\ \exists e \in E.\ e \sqsubseteq e'\}$, its downward closure is $E{\downarrow} = \{e' \in LExp_\perp |\ \exists e \in E.\ e' \sqsubseteq e\}$, and those operators are overloaded for let-expressions as $e{\uparrow} = \{e\}{\uparrow}$ and $e{\downarrow} = \{e\}{\downarrow}$.

Proof

1. Easily by induction on the structure of t.
2. Straightforward by induction on the structure of e. In the case of let expressions, the proof uses $|e| \in CTerm_\perp$ and Proposition 4 in order to apply the CRWL$_{let}$ rule (Let).
3. By Lemma 28 we have that $[\![e]\!] \Subset \lambda\theta.(|e\theta|{\uparrow}){\downarrow}$. By definition of hyperinclusion — Definition 8— we know that $[\![e]\!]\epsilon \subseteq (\lambda\theta.(|e\theta|{\uparrow}){\downarrow})\epsilon$, so $[\![e]\!]\epsilon = [\![e\epsilon]\!] \equiv [\![e]\!] \subseteq (|e|{\uparrow}){\downarrow} \equiv (|e\epsilon|{\uparrow}){\downarrow} = (\lambda\theta.(|e\theta|{\uparrow}){\downarrow})\epsilon$.
   □

Proposition 3 (Polarity of CRWL$_{let}$)
For any program $e, e' \in LExp_\perp$, $t, t' \in CTerm_\perp$, if $e \sqsubseteq e'$ and $t' \sqsubseteq t$ then $e \rightarrowtriangle t$ implies $e' \rightarrowtriangle t'$ with a proof of the same size or smaller—where the size of a CRWL$_{let}$-proof is measured as the number of rules of the calculus used in the proof.

Proof
By induction on the size of the CRWL-derivation. All the cases are straightforward except the (Let) rule:

(Let) We have the derivation:

$$\frac{e_1 \rightarrowtriangle t_1 \quad e_2[X/t_1] \rightarrowtriangle t}{e \equiv let\ X = e_1\ in\ e_2 \rightarrowtriangle t}\ (Let)$$

Since $e \sqsubseteq e'$ then $e' \equiv let\ X = e'_1\ in\ e'_2$ with $e_1 \sqsubseteq e'_1$ and $e_2 \sqsubseteq e'_2$. As $e_1 \sqsubseteq e'_1$ and $t_1 \sqsubseteq t_1$ —because $\sqsubseteq$ is reflexive— then by IH we have $e'_1 \rightarrowtriangle t_1$. We know that $e_2 \sqsubseteq e'_2$ so by Lemma 24 we have $e_2[X/t_1] \sqsubseteq e'_2[X/t_1]$ and by IH $\mathcal{P} \vdash_{CRWL_{let}} e'_2[X/t_1] \rightarrowtriangle t'$ such that $t' \sqsubseteq t$. Therefore:

$$\frac{e'_1 \rightarrowtriangle t_1 \quad e'_2[X/t_1] \rightarrowtriangle t'}{e' \equiv let\ X = e'_1\ in\ e'_2 \rightarrowtriangle t'}\ (Let)$$

□

Proposition 4 (Closedness under c-substitutions)
For any $e \in LExp_\perp$, $t \in CTerm_\perp$, $\theta \in CSubst_\perp$, $t \in [\![e]\!]$ implies $t\theta \in [\![e\theta]\!]$.

Proof
By induction on the size of the CRWL$_{let}$-proof. All the cases are straightforward except the (Let) rule:

(Let) In this case the expression is $e \equiv let\ X = e_1\ in\ e_2$ so we have a derivation

$$\frac{e_1 \rightarrowtriangle t_1 \quad e_2[X/t_1] \rightarrowtriangle t}{let\ X = e_1\ in\ e_2 \rightarrowtriangle t}\ (Let)$$

By IH we have that $e_1\theta \rightarrowtriangle t_1\theta$ and $(e_2[X/t_1])\theta \rightarrowtriangle t\theta$. By the variable convention we assume that $X \notin dom(\theta) \cup vran(\theta)$, so by Lemma 1 $e_2[X/t_1]\theta \equiv e_2\theta[X/t_1\theta]$ and $e_2\theta[X/t_1\theta] \rightarrowtriangle t\theta$. Then we can construct the proof:

$$\frac{e_1\theta \rightarrowtriangle t_1\theta \quad e_2\theta[X/t_1\theta] \rightarrowtriangle t\theta}{let\ X = e_1\theta\ in\ e_2\theta \rightarrowtriangle t\theta}\ (Let)$$

□

Theorem 5 (Weak Compositionality of CRWL$_{let}$)
For any $\mathcal{C} \in Cntxt$, $e \in LExp_\perp$

$$[\![\mathcal{C}[e]]\!] = \bigcup_{t \in [\![e]\!]} [\![\mathcal{C}[t]]\!] \qquad \text{if } BV(\mathcal{C}) \cap FV(e) = \emptyset$$

As a consequence, $[\![let\ X = e_1\ in\ e_2]\!] = \bigcup_{t_1 \in [\![e_1]\!]} [\![e_2[X/t_1]]\!]$.

Proof
We prove that $\mathcal{C}[e] \rightarrowtriangle t \Leftrightarrow \exists s \in CTerm_\perp$ such that $e \rightarrowtriangle s$ and $\mathcal{C}[s] \rightarrowtriangle t$.

$\Rightarrow$) By induction on the size of the proof for $\mathcal{C}[e] \rightarrowtriangle t$. The proof proceeds in a similar way to the proof for Theorem 1, page 53, so we only have to prove the (Let) case:

(Let) There are two cases depending on the context $\mathcal{C}$ (since $\mathcal{C} \neq [\ ]$):

- $\mathcal{C} \equiv let\ X = C'\ in\ e_2$) Straightforward.

- $\mathcal{C} \equiv let\ X = e_1\ in\ \mathcal{C}'$) The proof is

$$\frac{e_1 \rightarrowtriangle t_1 \quad \mathcal{C}'[e][X/t_1] \rightarrowtriangle t}{\mathcal{C}[e] \equiv let\ X = e_1\ in\ \mathcal{C}'[e] \rightarrowtriangle t}\ (Let)$$

  We assume that $X \notin var(t_1)$ by the variable convention, since X is bound in $\mathcal{C}$ and we can rename it freely. Moreover, we assume also that $X \notin BV(\mathcal{C}')$ because X is bound in $\mathcal{C}$, so we could rename the bound occurrences in $\mathcal{C}'$. Therefore $(dom([X/t_1] \cup vran([X/t_1])) \cap BV(\mathcal{C}') = \emptyset$ and $\mathcal{C}'[e][X/t_1] \equiv (\mathcal{C}'[X/t_1])[e[X/t_1]]$ by Lemma 25. Since $BV(\mathcal{C}) \cap FV(e) = \emptyset$ by the premise and $X \in BV(\mathcal{C})$ then $X \notin FV(e)$, so $(\mathcal{C}'[X/t_1])[e[X/t_1]] \equiv \mathcal{C}'[X/t_1][e]$. Then by IH $\exists s \in CTerm_\perp$ such that $e \rightarrowtriangle s$ and $\mathcal{C}'[X/t_1][s] \rightarrowtriangle t$. Therefore we can build:

$$\frac{e_1 \rightarrowtriangle t_1 \quad \mathcal{C}'[s][X/t_1] \equiv^{(*)} \mathcal{C}'[X/t_1][s] \rightarrowtriangle t}{\mathcal{C}[s] \equiv let\ X = e_1\ in\ \mathcal{C}'[s] \rightarrowtriangle t}\ (Let)$$

  (*) Using Lemma 25 as above and the assumption that $X \notin var(s)$ by the variable convention, since X is bound in $\mathcal{C}$ and we can rename it freely.

$\Leftarrow$) By induction on the size of the proof for $\mathcal{C}[s] \rightarrowtriangle t$. As before, the proof proceeds in a similar way to the proof for Theorem 1, page 53, so we only have to prove the (Let) case:

(Let) If we use (Let) then there are two cases depending on the context $\mathcal{C}$ (since $\mathcal{C} \neq [\ ]$):

- $\mathcal{C} = let\ X = \mathcal{C}'\ in\ e_2$) Straighforward.
- $\mathcal{C} = let\ X = e_1\ in\ \mathcal{C}'$) then we have $e \rightarrowtriangle s$ and

$$\frac{e_1 \rightarrowtriangle t_1 \quad \mathcal{C}'[s][X/t_1] \rightarrowtriangle t}{\mathcal{C}[s] \equiv let\ X = e_1\ in\ \mathcal{C}'[s] \rightarrowtriangle t}\ (Let)$$

  By the same reasoning as in the second case of the (Let) rule of the $\Rightarrow$) part of this theorem, $\mathcal{C}'[s][X/t_1] \equiv \mathcal{C}'[X/t_1][s]$. Then by IH $\mathcal{C}'[X/t_1][e] \rightarrowtriangle t$. Again by the same reasoning we have $\mathcal{C}'[e][X/t_1] \equiv \mathcal{C}'[X/t_1][e]$, so we can build the proof:

$$\frac{e_1 \rightarrowtriangle t_1 \quad \mathcal{C}'[e][X/t_1] \equiv \mathcal{C}'[X/t_1][e] \rightarrowtriangle t}{\mathcal{C}[e] \equiv let\ X = e_1\ in\ \mathcal{C}'[e] \rightarrowtriangle t}\ (Let)$$

This ends the proof of the main part of the theorem. With respect to the consequence $[\![let\ X = e_1\ in\ e_2]\!]_{CRWL_{let}} = \bigcup_{t_1 \in [\![e_1]\!]_{CRWL_{let}}} [\![e_2[X/t_1]]\!]_{CRWL_{let}}$ we have:

$$\begin{array}{ll} [\![let\ X = e_1\ in\ e_2]\!]_{CRWL_{let}} & \\ = [\![(let\ X = [\ ]\ in\ e_2)[e_1]]\!]_{CRWL_{let}} & \\ = \bigcup\limits_{t_1 \in [\![e_1]\!]_{CRWL_{let}}} [\![let\ X = t_1\ in\ e_2]\!]_{CRWL_{let}} & \text{by Theorem 5} \\ = \bigcup\limits_{t_1 \in [\![e_1]\!]_{CRWL_{let}}} [\![e_2[X/t_1]]\!]_{CRWL_{let}} & \text{by Proposition 8} \end{array}$$

In the last step we replace $let\ X = t_1\ in\ e_2$ by $e_2[X/t_1]$ which is a (Bind) step of $\rightarrow^{lnf}$, so by Proposition 8 it preserves the denotation. □

For Proposition 5, in this Appendix we prove a generalization of the statement appearing in Section 4.2 (page 21). However, it is easy to check that Proposition 5 in Section 4.2 follows easily from points *2* and *3* here.

Proposition 5 (Monotonicity for substitutions of $CRWL_{let}$)
For any program $e \in LExp_\perp$, $t \in CTerm_\perp$, $\sigma, \sigma' \in LSubst_\perp$

1. If $\forall X \in \mathcal{V}, s \in CTerm_\perp$ given $\sigma(X) \rightarrowtriangle s$ with size K we also have $\sigma'(X) \rightarrowtriangle s$ with size $K' \leq K$, then $e\sigma \rightarrowtriangle t$ with size L implies $e\sigma' \rightarrowtriangle t$ with size $L' \leq L$.
2. If $\sigma \sqsubseteq \sigma'$ then $e\sigma \rightarrowtriangle t$ implies $e\sigma' \rightarrowtriangle t$ with a proof of the same size or smaller.
3. If $\sigma \trianglelefteq \sigma'$ then $[\![e\sigma]\!] \subseteq [\![e\sigma']\!]$.

Proof

1. If $e \equiv X \in \mathcal{V}$, assume $X\sigma \rightarrowtriangle t$, then $X\sigma' \rightarrowtriangle t$ with a proof of the same size or smaller, by hypothesis. Otherwise we proceed by induction on the structure of the proof $e\sigma \rightarrowtriangle t$.

Base cases

(B) Then $t \equiv \perp$ and $e\sigma' \rightarrowtriangle \perp$ with a proof of size 1 just applying rule (B).

(RR) Then $e \in \mathcal{V}$ and we are in the previous case.

(DC) Then $e \equiv c \in CS^0$, as $e \notin \mathcal{V}$, hence $e\sigma \equiv c \equiv e\sigma'$ and every proof for $e\sigma \rightarrowtriangle t$ is a proof for $e\sigma' \rightarrowtriangle t$.

Inductive steps

(DC) Then $e \equiv c(e_1, \ldots, e_n)$, as $e \notin \mathcal{V}$, and we have:

$$\frac{e_1\sigma \rightarrowtriangle t_1 \quad \ldots \quad e_n\sigma \rightarrowtriangle t_n}{e\sigma \equiv c(e_1\sigma, \ldots, e_n\sigma) \rightarrowtriangle c(t_1, \ldots, t_n) \equiv t}\ (DC)$$

By IH or the proof of the other cases $\forall i \in \{1, \ldots, n\}$ we have $e_i\sigma' \rightarrowtriangle t_i$ with a proof of the same size or smaller, so we can built a proof for $e\sigma' \equiv c(e_1\sigma', \ldots, e_n\sigma') \rightarrowtriangle c(t_1, \ldots, t_n) \equiv t$ using (DC), with a size equal or smaller than the size of the starting proof.

(OR) Similar to the previous case.

(Let) Then $e \equiv let\ X = e_1\ in\ e_2$, as $e \notin \mathcal{V}$, and we have:

$$\frac{e_1\sigma \rightarrowtriangle t_1 \quad e_2\sigma[X/t_1] \rightarrowtriangle t}{let\ X = e_1\sigma\ in\ e_2\sigma \rightarrowtriangle t}\ (Let)$$

By IH we have $e_1\sigma \rightarrowtriangle t_1$. By the variable convention we assume that $X \notin$

$dom(\sigma) \cup vran(\sigma)$ and $X \notin dom(\sigma') \cup vran(\sigma')$. Then it is easy to check that $\forall Y \in \mathcal{V}, s, t \in CTerm_\perp$, given $Y(\sigma[X/t]) \twoheadrightarrow s$ with size K we also have $Y(\sigma'[X/t]) \twoheadrightarrow s$ with size $K' \leq K$. Then by IH we have $e_2\sigma'[X/t_1] \twoheadrightarrow t$. Therefore we can construct a proof with a size equal or smaller than the starting one:

$$\frac{e_1\sigma' \twoheadrightarrow t_1 \quad e_2\sigma'[X/t_1] \twoheadrightarrow t}{let\ X = e_1\sigma'\ in\ e_2\sigma' \twoheadrightarrow t}\ (Let)$$

2. By induction on the size of the CRWL$_{let}$-proof. The cases for classical CRWL appear in (Vado-Vírseda 2002), so we only have to prove the case for the (Let) rule:

(Let) In this case the expression is $e \equiv let\ X = e_1\ in\ e_2$ so we have a proof

$$\frac{e_1\sigma \twoheadrightarrow t_1 \quad e_2\sigma[X/t_1] \twoheadrightarrow t}{let\ X = e_1\sigma\ in\ e_2\sigma \twoheadrightarrow t}\ (Let)$$

By IH we have that $e_1\sigma \twoheadrightarrow t_1$. By the variable convention we can assume that $BV(e) \cap (dom(\sigma) \cup vran(\sigma)) = \emptyset$ and $BV(e) \cap (dom(\sigma') \cup vran(\sigma')) = \emptyset$. With the previous properties it is easy to see that $\sigma[X/t_1] \sqsubseteq \sigma'[X/t_1]$, so by IH $e_2\sigma'[X/t_1] \twoheadrightarrow t$. Therefore we can build the proof:

$$\frac{e_1\sigma' \twoheadrightarrow t_1 \quad e_2\sigma'[X/t_1] \twoheadrightarrow t}{let\ X = e_1\sigma'\ in\ e_2\sigma' \twoheadrightarrow t}\ (Let)$$

3. By induction on the structure of e:

$e \equiv X \in \mathcal{V}$ **-** In this case $[\![X\sigma]\!]_{CRWL_{let}} \subseteq [\![X\sigma']\!]_{CRWL_{let}}$ because by the hypothesis $\sigma \trianglelefteq \sigma'$.

$e \equiv h(e_1, \ldots, e_n)$ **-** Applying Theorem 5 with $\mathcal{C} \equiv h([\ ], e_2\sigma, \ldots, e_n\sigma)$ we have $[\![h(e_1\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}} = [\![\mathcal{C}[e_1\sigma]]\!]_{CRWL_{let}} = \bigcup_{t \in [\![e_1\sigma]\!]_{CRWL_{let}}} [\![\mathcal{C}[t]]\!]_{CRWL_{let}}$ because $BV(\mathcal{C}) = \emptyset$. On the other hand, by Theorem 5 we also know that

$$\begin{array}{rcl} [\![h(e_1\sigma', e_2\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}} & = & [\![\mathcal{C}[e_1\sigma']]\!]_{CRWL_{let}} \\ & = & \bigcup_{t \in [\![e_1\sigma']\!]_{CRWL_{let}}} [\![\mathcal{C}[t]]\!]_{CRWL_{let}} \end{array}$$

Since by IH we have $[\![e_1\sigma]\!]_{CRWL_{let}} \subseteq [\![e_1\sigma']\!]_{CRWL_{let}}$ it is easy to check that

$$\bigcup_{t \in [\![e_1\sigma]\!]_{CRWL_{let}}} [\![\mathcal{C}[t]]\!]_{CRWL_{let}} \subseteq \bigcup_{t \in [\![e_1\sigma']\!]_{CRWL_{let}}} [\![\mathcal{C}[t]]\!]_{CRWL_{let}}$$

so $[\![h(e_1\sigma, e_2\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}} \subseteq [\![h(e_1\sigma', e_2\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}}$. Using the same reasoning in the rest of subexpressions $e_i\sigma$ we can prove:
$[\![h(e_1\sigma', e_2\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}} \subseteq [\![h(e_1\sigma', e_2\sigma', e_3\sigma \ldots, e_n\sigma)]\!]_{CRWL_{let}}$
$[\![h(e_1\sigma', e_2\sigma', e_3\sigma \ldots, e_n\sigma)]\!]_{CRWL_{let}} \subseteq [\![h(\ldots, e_3\sigma', e_4\sigma \ldots, e_n\sigma)]\!]_{CRWL_{let}}$
$\ldots$
$[\![\ldots, e_{n-1}\sigma', e_n\sigma)]\!]_{CRWL_{let}} \subseteq [\![h(e_1\sigma', \ldots, e_n\sigma')]\!]_{CRWL_{let}}$
Then by transitivity of $\subseteq$ we have:
$[\![h(e_1, \ldots, e_n)\sigma]\!]_{CRWL_{let}} \equiv [\![h(e_1\sigma, \ldots, e_n\sigma)]\!]_{CRWL_{let}} \subseteq$
$[\![h(e_1\sigma', \ldots, e_n\sigma')]\!]_{CRWL_{let}} \equiv [\![h(e_1, \ldots, e_n)\sigma']\!]_{CRWL_{let}}$.

$e \equiv let\ X = e_1\ in\ e_2$ - As Theorem 5 states, $[\![let\ X = e_1\sigma\ in\ e_2\sigma]\!]_{CRWL_{let}} = \bigcup_{t_1 \in [\![e_1\sigma]\!]_{CRWL_{let}}} [\![e_2\sigma[X/t_1]]\!]_{CRWL_{let}}$.

By the Induction Hypothesis we have that $[\![e_1\sigma]\!]_{CRWL_{let}} \subseteq [\![e_1\sigma']\!]_{CRWL_{let}}$. Due to the variable convention we assume that $X \notin dom(\sigma) \cup vran(\sigma)$ and $X \notin dom(\sigma') \cup vran(\sigma')$, so it is easy to check that $\sigma[X/t] \trianglelefteq \sigma'[X/t]$ for any $t \in CTerm$. Then by the Induction Hypothesis we know that $[\![e_2\sigma[X/t]]\!]_{CRWL_{let}} \subseteq [\![e_2\sigma'[X/t]]\!]_{CRWL_{let}}$. Therefore

$$\begin{array}{rcl}
[\![(let\ X = e_1\ in\ e_2)\sigma]\!]_{CRWL_{let}} & = & \bigcup_{t_1 \in [\![e_1\sigma]\!]_{CRWL_{let}}} [\![e_2\sigma[X/t_1]]\!]_{CRWL_{let}} \\
& \subseteq & \bigcup_{t_1 \in [\![e_1\sigma']\!]_{CRWL_{let}}} [\![e_2\sigma'[X/t_1]]\!]_{CRWL_{let}} \\
& = & [\![let\ X = e_1\sigma'\ in\ e_2\sigma']\!]_{CRWL_{let}} \\
& = & [\![(let\ X = e_1\ in\ e_2)\sigma']\!]_{CRWL_{let}}
\end{array}$$

□

Theorem 6 (Compositionality of hypersemantics)
For all $\mathcal{C} \in Cntxt$, $e \in LExp_\perp$

$$[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}]\!][\![e]\!]$$

As a consequence: $[\![e]\!] = [\![e']\!] \Leftrightarrow \forall \mathcal{C} \in Cntxt.[\![\mathcal{C}[e]]\!] = [\![\mathcal{C}[e']]\!]$.

Proof
By induction over the structure of contexts. The base case is $\mathcal{C} = []$, so $[\![\mathcal{C}[e]]\!] = [\![e]\!] = [\![[]]\!][\![e]\!] = [\![\mathcal{C}]\!][\![e]\!]$, as $[\![[]]\!]$ is the identity function by definition. Regarding the inductive step:

- $\mathcal{C} = h(e_1, \ldots, \mathcal{C}', \ldots, e_n)$: Then

$$\begin{array}{ll}
[\![\mathcal{C}]\!][\![e]\!] = \lambda\theta. \bigcup_{t \in [\![\mathcal{C}']\!][\![e]\!]\theta} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \\
= \lambda\theta. \bigcup_{t \in [\![\mathcal{C}'[e]]\!]\theta} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \text{by IH} \\
= \lambda\theta. \bigcup_{t \in [\![(\mathcal{C}'[e])\theta]\!]} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \text{by definition} \\
= \lambda\theta.[\![h(e_1\theta, \ldots, (\mathcal{C}'[e])\theta, \ldots, e_n\theta)]\!] & \text{by Lemma 5} \\
= \lambda\theta.[\![(\mathcal{C}[e])\theta]\!] = [\![\mathcal{C}[e]]\!] &
\end{array}$$

- $\mathcal{C} = let\ X = \mathcal{C}'\ in\ s$: Then

$$\begin{array}{ll}
[\![\mathcal{C}]\!][\![e]\!] = \lambda\theta. \bigcup_{t \in [\![\mathcal{C}']\!][\![e]\!]\theta} [\![let\ X = t\ in\ s\theta]\!] & \text{by definition} \\
= \lambda\theta. \bigcup_{t \in [\![\mathcal{C}']\!][\![e]\!]\theta} [\![s\theta[X/t]]\!] & \text{by rule (Bind)}^{(*)} \\
= \lambda\theta. \bigcup_{t \in [\![\mathcal{C}'[e]]\!]\theta} [\![s\theta[X/t]]\!] & \text{by IH} \\
= \lambda\theta. \bigcup_{t \in [\![(\mathcal{C}'[e])\theta]\!]} [\![s\theta[X/t]]\!] & \text{by definition} \\
= \lambda\theta.[\![let\ X = (\mathcal{C}'[e])\theta\ in\ s\theta]\!] & \text{by Lemma 5} \\
= [\![\mathcal{C}[e]]\!] &
\end{array}$$

(*): by Proposition 8 $[\![let\ X = t\ in\ s\theta]\!] = [\![s\theta[X/t]]\!]$ since $let\ X = t\ in\ s\theta \to^{lnf} s\theta[X/t]$.

- $\mathcal{C} = let\ X = s\ in\ \mathcal{C}'$: Then

$$\begin{aligned}
&\llbracket\mathcal{C}\rrbracket\llbracket e\rrbracket = \lambda\theta. \bigcup_{t\in\llbracket s\rrbracket\theta} \llbracket\mathcal{C}'\rrbracket\llbracket e\rrbracket(\theta[X/t]) && \\
&= \lambda\theta. \bigcup_{t\in\llbracket s\rrbracket\theta} \llbracket\mathcal{C}'[e]\rrbracket(\theta[X/t]) && \text{by IH} \\
&= \lambda\theta. \bigcup_{t\in\llbracket s\rrbracket\theta} \llbracket(\mathcal{C}'[e])(\theta[X/t])\rrbracket && \text{by definition} \\
&= \lambda\theta. \bigcup_{t\in\llbracket s\theta\rrbracket} \llbracket(\mathcal{C}'[e])(\theta[X/t])\rrbracket && \text{by definition} \\
&= \lambda\theta. \bigcup_{t\in\llbracket s\theta\rrbracket} \llbracket((\mathcal{C}'[e])\theta)[X/t]\rrbracket && \\
&= \lambda\theta.\llbracket let\ X = s\theta\ in\ (\mathcal{C}'[e])\theta\rrbracket && \text{by Lemma 5} \\
&= \llbracket\mathcal{C}[e]\rrbracket &&
\end{aligned}$$

□

Proposition 6

Consider two sets A, B, and let $\mathcal{F}$ be the set of functions $A \to \mathcal{P}(B)$. Then:

i) $\Subset$ is indeed a partial order on $\mathcal{F}$, and Δf is indeed a decomposition of $f \in \mathcal{F}$, i.e., $\mathop{\Cup}\ (\Delta f) = f$.

ii) Monotonicity of hyperunion wrt. inclusion: for any $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{F}$

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \text{ implies } \mathop{\Cup}\ \mathcal{I}_1 \Subset \mathop{\Cup}\ \mathcal{I}_2$$

iii) Distribution of unions: for any $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{F}$

$$\mathop{\Cup}\ (\mathcal{I}_1 \cup \mathcal{I}_2) = (\mathop{\Cup}\ \mathcal{I}_1) \Cup (\mathop{\Cup}\ \mathcal{I}_2)$$

iv) Monotonicity of decomposition wrt. hyperinclusion: for any $f_1, f_2 \in \mathcal{F}$

$$f_1 \Subset f_2 \text{ implies } \Delta f_1 \subseteq \Delta f_2$$

Proof

i) The binary relation $\Subset$ is a partial order on $\mathcal{F}$ because:

- It is reflexive, as for any function f and any $x \in A$ we have that $f(x) = f(x)$, and thus $f(x) \subseteq f(x)$, therefore $f \Subset f$.
- It is transitive because given some functions f_1, f_2, f_3 such that $f_1 \Subset f_2$ and $f_2 \Subset f_3$, then for any $x \in A$ we have $f_1(x) \subseteq f_2(x) \subseteq f_3(x)$ by definition of $\Subset$, hence $f_1 \Subset f_3$.
- It is antisymmetric wrt. extensional function equality, because for any pair of hypersemantics f_1, f_2 such that $f_1 \Subset f_2$ and $f_2 \Subset f_1$ and any $x \in A$ we have that $f_1(x) \subseteq f_2(x)$ and $f_2(x) \subseteq f_1(x)$ by definition of $\Subset$, hence $f_1(x) = f_2(x)$ by antisymmetry of $\subseteq$ and $f_1 = f_2$.

In order to prove that Δf is indeed a decomposition of $f \in \mathcal{F}$ we first perform a little massaging by using the definitions of $\mathop{\Cup}$ and Δ.

$$\mathop{\Cup}\ (\Delta f) = \mathop{\Cup}\ \{\hat{\lambda} a.\{b\} \mid a \in A, b \in f(a)\} = \lambda x \in A. \bigcup_{a\in A} \bigcup_{b\in f(a)} (\hat{\lambda} a.\{b\})x$$

Now we will use the fact that $\Subset$ is a partial order, and therefore it is antisymmetric, so mutual inclusion by $\Subset$ implies equality.

- $\underline{f \Subset \Cup\ (\Delta f)}$: Given arbitraries $a \in A$, $b \in f(a)$ then

$$\begin{array}{ll}
(\Cup\ (\Delta f))a = \bigcup\limits_{x \in A} \bigcup\limits_{y \in f(x)} (\hat{\lambda} x.\{y\})a & \\
\supseteq \bigcup\limits_{y \in f(a)} (\hat{\lambda} a.\{y\})a & \text{as } a \in A \\
= \bigcup\limits_{y \in f(a)} \{y\} \ni b & \text{as } b \in f(a)
\end{array}$$

- $\underline{\Cup\ (\Delta f) \Subset f}$: Given arbitraries $a \in A$, $b \in (\Cup\ (\Delta f))a$ then we have that $b \in \bigcup\limits_{x \in A} \bigcup\limits_{y \in f(x)} (\hat{\lambda} x.\{y\})a$, therefore $\exists x \in A, y \in f(x)$ such that $b \in (\hat{\lambda} x.\{y\})a$. But then $a \equiv x$ —otherwise $(\hat{\lambda} x.\{y\})a = \emptyset$— and $y \equiv b$ —because $b \in (\hat{\lambda} x.\{y\})a = \{y\}$—, and so $y \in f(x)$ implies $b \in f(a)$.

ii) Given an arbitrary $a \in A$ then

$$\begin{array}{ll}
(\Cup\ \mathcal{I}_1)a = \bigcup\limits_{f \in \mathcal{I}_1} f(a) & \text{by definition of } \Cup \\
\subseteq \bigcup\limits_{f \in \mathcal{I}_2} f(a) & \text{as } \mathcal{I}_1 \subseteq \mathcal{I}_2 \\
= (\Cup\ \mathcal{I}_2)a & \text{by definition of } \Cup
\end{array}$$

iii)

$$\begin{array}{ll}
\Cup\ (\mathcal{I}_1 \cup \mathcal{I}_2) = \lambda a. \bigcup\limits_{f \in (\mathcal{I}_1 \cup \mathcal{I}_2)} f(a) & \text{by definition of } \Cup \\
= \lambda a. \bigcup\limits_{f \in \mathcal{I}_1} f(a) \ \cup \bigcup\limits_{f \in \mathcal{I}_2} f(a) & \\
= \lambda a.(\Cup\ \mathcal{I}_1)a \cup (\Cup\ \mathcal{I}_2)a & \text{by definition of } \Cup \\
= (\Cup\ \mathcal{I}_1) \Cup\ (\Cup\ \mathcal{I}_2) & \text{by definition of } \Cup
\end{array}$$

iv) Suppose an arbitrary $\hat{\lambda} a.\{b\} \in \Delta f_1$ with $a \in A$ and $b \in f_1(a)$ by definition. Since $f_1 \Subset f_2$ then $f_1(a) \subseteq f_2(a)$. Therefore $b \in f_2(a)$ and $\hat{\lambda} a.\{b\} \in \Delta f_2$.

□

Proposition 7 (Distributivity under context of hypersemantics union)

$$\llbracket \mathcal{C} \rrbracket (\Cup\ H) = \mathop{\Cup}\limits_{\varphi \in H} \llbracket \mathcal{C} \rrbracket \varphi$$

Proof

We proceed by induction on the structure of $\mathcal{C}$. Regarding the base case, then $\mathcal{C} = []$ and so:

$$\begin{array}{ll}
\llbracket \mathcal{C} \rrbracket (\Cup\ H) = \Cup\ H & \text{by definition of } \llbracket \mathcal{C} \rrbracket \\
= \mathop{\Cup}\limits_{\varphi\ \in\ H} \varphi & \\
= \mathop{\Cup}\limits_{\varphi\ \in\ H} \llbracket \mathcal{C} \rrbracket \varphi & \text{by definition of } \llbracket \mathcal{C} \rrbracket
\end{array}$$

For the inductive step we have several possibilities.

- $\mathcal{C} \equiv h(e_1, \ldots, \mathcal{C}', \ldots, e_n)$: then

$$\begin{array}{ll}
[\![\mathcal{C}]\!](\Cup\ H) = \lambda\theta. \displaystyle\bigcup_{t \in [\![\mathcal{C}']\!](\Cup\ H)\theta} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \text{by definition of } [\![\mathcal{C}]\!] \\
= \lambda\theta. \displaystyle\bigcup_{t \in ((\Cup\ \{[\![\mathcal{C}']\!]\varphi \ \mid\ \varphi \in H\})\theta)} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \text{by IH} \\
= \lambda\theta. \displaystyle\bigcup_{t \in (\bigcup_{\varphi \in H} [\![\mathcal{C}']\!]\varphi\theta)} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \text{by definition of } \Cup \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} \bigcup_{t \in [\![\mathcal{C}']\!]\varphi\theta} [\![h(e_1\theta, \ldots, t, \ldots, e_n\theta)]\!] & \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} [\![\mathcal{C}]\!]\varphi\theta & \text{by definition of } [\![\mathcal{C}]\!] \\
= \displaystyle\Cup_{\varphi\ \in\ H} [\![\mathcal{C}]\!]\varphi & \text{by definition of } \Cup
\end{array}$$

- $\mathcal{C} \equiv let\ X = \mathcal{C}'\ in\ e$: then

$$\begin{array}{ll}
[\![\mathcal{C}]\!](\Cup\ H) = \lambda\theta. \displaystyle\bigcup_{t \in [\![\mathcal{C}']\!](\Cup\ H)\theta} [\![let\ X = t\ in\ e\theta]\!] & \text{by definition of } [\![\mathcal{C}]\!] \\
= \lambda\theta. \displaystyle\bigcup_{t \in ((\Cup\ \{[\![\mathcal{C}']\!]\varphi \ \mid\ \varphi \in H\})\theta)} [\![let\ X = t\ in\ e\theta]\!] & \text{by IH} \\
= \lambda\theta. \displaystyle\bigcup_{t \in (\bigcup_{\varphi \in H} [\![\mathcal{C}']\!]\varphi\theta)} [\![let\ X = t\ in\ e\theta]\!] & \text{by definition of } \Cup \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} \bigcup_{t \in [\![\mathcal{C}']\!]\varphi\theta} [\![let\ X = t\ in\ e\theta]\!] & \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} [\![\mathcal{C}]\!]\varphi\theta & \text{by definition of } [\![\mathcal{C}]\!] \\
= \displaystyle\Cup_{\varphi\ \in\ H} [\![\mathcal{C}]\!]\varphi & \text{by definition of } \Cup
\end{array}$$

- $\mathcal{C} \equiv let\ X = e\ in\ \mathcal{C}'$: then

$$\begin{array}{ll}
[\![\mathcal{C}]\!](\Cup\ H) = \lambda\theta. \displaystyle\bigcup_{t \in [\![e]\!]\theta} [\![\mathcal{C}']\!](\Cup\ H)(\theta[X/t]) & \text{by definition of } [\![\mathcal{C}]\!] \\
= \lambda\theta. \displaystyle\bigcup_{t \in [\![e]\!]\theta} (\Cup\ \{[\![\mathcal{C}']\!]\varphi \mid \varphi \in H\})(\theta[X/t]) & \text{by IH} \\
= \lambda\theta. \displaystyle\bigcup_{t \in [\![e]\!]\theta} \bigcup_{\varphi \in H} [\![\mathcal{C}']\!]\varphi(\theta[X/t]) & \text{by definition of } \Cup \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} \bigcup_{t \in [\![e]\!]\theta} [\![\mathcal{C}']\!]\varphi(\theta[X/t]) & \text{as } H \text{ is independent from } t \\
= \lambda\theta. \displaystyle\bigcup_{\varphi \in H} [\![\mathcal{C}]\!]\varphi\theta & \text{by definition of } [\![\mathcal{C}]\!] \\
= \displaystyle\Cup_{\varphi\ \in\ H} [\![\mathcal{C}]\!]\varphi & \text{by definition of } \Cup
\end{array}$$

□

A.7 Proofs for Section 4.3

Theorem 9 (Hyper-Soundness of let-rewriting)
For all $e, e' \in LExp$, if $e \to^{l^*} e'$ then $[\![e']\!] \Subset [\![e]\!]$.

Proof
We first prove the theorem for a single step of $\to^{l}$. We proceed assumming some

$\theta \in CSubst_\perp$ such that $e'\theta \rightarrowtriangle t$ and then proving $e\theta \rightarrowtriangle t$. The case where $t \equiv \perp$ holds trivially using the rule **B**, so we will prove the rest by a case distinction on the rule of the let-rewriting calculus applied:

(Fapp) Assume $f(t_1, \ldots, t_n) \to^l r$ with $(f(p_1, \ldots, p_n) \to e) \in \mathcal{P}$, $\sigma \in CSubst$, such that $\forall i.p_i\sigma \equiv t_i$ and $e\sigma \equiv r$, and $\theta \in CSubts_\perp$ such that $r\theta \rightarrowtriangle t$. Then as $\sigma\theta \in CSubts_\perp, \forall i.p_i\sigma\theta \equiv t_i\theta$ and $e\sigma\theta \equiv r\theta$ we can use the (OR) rule to build the following proof:

$$\dfrac{\dfrac{\text{Lemma 18}}{t_1\theta \rightarrowtriangle t_1\theta} \quad \ldots \quad \dfrac{\text{Lemma 18}}{t_n\theta \rightarrowtriangle t_n\theta} \quad r\theta \rightarrowtriangle t}{f(t_1\theta, \ldots, t_n\theta) \rightarrowtriangle t}\ (OR)$$

(LetIn) Assume $h(\ldots, e, \ldots) \to^l let\ X = e\ in\ h(\ldots, X, \ldots)$ by (LetIn) and $\theta \in CSubts_\perp$ such that $(let\ X = e\ in\ h(\ldots, X, \ldots))\theta \rightarrowtriangle t$. This proof must be of the shape of:

$$\dfrac{e\theta \rightarrowtriangle t_1 \qquad h(d_1\theta, \ldots, X\theta, \ldots, d_n\theta)[X/t_1] \rightarrowtriangle t}{let\ X = e\theta\ in\ h(d_1\theta, \ldots, X\theta, \ldots, d_n\theta) \rightarrowtriangle t}\ (Let)$$

for some $d_1, \ldots, d_n \in LExp, t_1 \in CTerm_\perp$. Besides $X \notin (dom(\theta) \cup vran(\theta))$ by the variable convention[5], hence $X\theta \equiv X$ and so $h(d_1\theta, \ldots, X\theta, \ldots, d_n\theta)[X/t_1] \equiv h(d_1\theta, \ldots, t_1, \ldots, d_n\theta)$, as X is fresh by the conditions in (LetIn) and so it does not appear in any d_i. Now we have two possibilities:

a) $h \equiv c \in DC$: Then $h(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle t$ must proved by (DC):

$$\dfrac{d_1\theta \rightarrowtriangle s_1 \ \ldots\ t_1 \rightarrowtriangle t'_1 \ \ldots\ d_n\theta \rightarrowtriangle s_n}{c(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle c(s_1, \ldots, t'_1, \ldots, s_n) \equiv t}\ (DC)$$

for some $s_1, \ldots, s_n, t'_1 \in CTerm_\perp$. Then $t_1 \rightarrowtriangle t'_1$ implies $t'_1 \sqsubseteq t_1$ by Lemma 5, hence $e\theta \rightarrowtriangle t_1$ implies $e\theta \rightarrowtriangle t'_1$ by Proposition 3, and we can build the following proof:

$$\dfrac{d_1\theta \rightarrowtriangle s_1 \ \ldots\ e\theta \rightarrowtriangle t'_1 \ \ldots\ d_n\theta \rightarrowtriangle s_n}{h(\ldots, e, \ldots)\theta \equiv c(d_1\theta, \ldots, e\theta, \ldots, d_n\theta) \rightarrowtriangle c(s_1, \ldots, t'_1, \ldots, s_n) \equiv t}$$

b) $h \equiv f \in FS$: Then $h(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle t$ must be proved by (OR):

$$\dfrac{d_1\theta \rightarrowtriangle s_1\sigma \quad \ldots \quad t_1 \rightarrowtriangle t'_1\sigma \quad \ldots \quad d_n\theta \rightarrowtriangle s_n\sigma \quad r\sigma \rightarrowtriangle t}{f(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle t}\ (OR)$$

for some $s_1\sigma, \ldots, s_n\sigma, t'_1\sigma \in CTerm_\perp$, $(f(s_1, \ldots, t'_1, \ldots s_n) \to r) \in \mathcal{P}$, $\sigma \in CSubst_\perp$. Then we can prove $e\theta \rightarrowtriangle t'_1\sigma$ like in the previous case, to build the following proof:

$$\dfrac{d_1\theta \rightarrowtriangle s_1\sigma \quad \ldots \quad e\theta \rightarrowtriangle t'_1\sigma \quad \ldots \quad d_n\theta \rightarrowtriangle s_n\sigma \quad r\sigma \rightarrowtriangle t}{h(\ldots, e, \ldots)\theta \equiv f(d_1\theta, \ldots, e\theta, \ldots, d_n\theta) \rightarrowtriangle t}\ (OR)$$

[5] Actually, to prove this theorem properly, we cannot restrict the substitution to fulfill these restrictions, so in fact we rename the bound variables in an α-conversion fashion and use the equivalence $e[X/e'] \equiv e[X/Y][Y/e']$ (with Y the new bound variable), to use the hypothesis. This will be done implicitly when needed during the remaining of the proof.

(Bind) Assume $let\ X = t_1\ in\ e \to^l e[X/t_1]$ by (Bind) and $\theta \in CSubst_\perp$ such that $(e[X/t_1])\theta \rightarrowtriangle t$. Then $X \notin (dom(\theta) \cup vran(\theta))$ by the variable convention, so we can apply Lemma 1 (Substitution lemma) to get $e\theta[X/t_1\theta] \equiv (e[X/t_1])\theta$. Besides $t_1 \in CTerm$ and $\theta \in CSubst_\perp$ by hypothesis, hence $t_1\theta \in CTerm_\perp$ and we can build the following proof:

$$\dfrac{\dfrac{\text{Lemma 18}}{t_1\theta \rightarrowtriangle t_1\theta} \qquad e\theta[X/t_1\theta] \equiv (e[X/t_1])\theta \rightarrowtriangle t}{let\ X = t_1\theta\ in\ e\theta \rightarrowtriangle t}\ (Let)$$

(Elim) Assume $let\ X = e_1\ in\ e_2 \to^l e_2$ by (Elim) and $\theta \in CSubts_\perp$ such that $e_2\theta \rightarrowtriangle t$. Then $X \notin vran(\theta)$ by the variable convention and $X \notin FV(e_2)$ by the condition of (Elim), hence $e_2\theta[X/\perp] \equiv e_2\theta$ and we can build the following proof:

$$\dfrac{\dfrac{}{e_1\theta \rightarrowtriangle \perp}\ (B) \qquad e_2\theta[X/\perp] \equiv e_2\theta \rightarrowtriangle t}{let\ X = e_1\theta\ in\ e_2\theta \rightarrowtriangle t}\ (Let)$$

(Flat) Assume $let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \to^l let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3)$ by (Flat) and $\theta \in CSubts_\perp$ such that $(let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3))\theta \rightarrowtriangle t$. This proof must be must be of the shape of:

$$\dfrac{e_1\theta \rightarrowtriangle t_1 \qquad \dfrac{e_2\theta[Y/t_1] \rightarrowtriangle t_2 \quad e_3\theta[Y/t_1][X/t_2] \rightarrowtriangle t}{(let\ X = e_2\theta\ in\ e_3\theta)[Y/t_1] \rightarrowtriangle t}\ (Let)}{let\ Y = e_1\theta\ in\ (let\ X = e_2\theta\ in\ e_3\theta) \rightarrowtriangle t}\ (Let)$$

for some $t_1, t_2 \in CTerm_\perp$. Besides $Y \notin vran(\theta)$ by the variable convention and $Y \notin FV(e_3)$ by the condition of (Flat), hence $e_3\theta[Y/t_1] \equiv e_3\theta$ and we can build the following proof:

$$\dfrac{\dfrac{\dfrac{\mathit{Hypothesis}}{e_1\theta \rightarrowtriangle t_1} \qquad \dfrac{\mathit{Hypothesis}}{e_2\theta[Y/t_1] \rightarrowtriangle t_2}}{let\ Y = e_1\theta\ in\ e_2\theta \rightarrowtriangle t_2}\ (Let) \qquad e_3\theta[X/t_2] \equiv e_3\theta[Y/t_1][X/t_2] \rightarrowtriangle t}{let\ X = (let\ Y = e_1\theta\ in\ e_2\theta)\ in\ e_3\theta \rightarrowtriangle t}\ (Let)$$

(Contx) By the proof of the other cases, $\llbracket e' \rrbracket \Subset \llbracket e \rrbracket$, but then $\llbracket \mathcal{C}[e'] \rrbracket \Subset \llbracket \mathcal{C}[e] \rrbracket$ by Lemma 7, and we are done.

The proof for several steps is a trivial induction on the length of the derivation $e \to^{l^*} e'$. $\square$

Proposition 8 (The $\to^{lnf}$ relation preserves hyperdenotation)
For all $e, e' \in LExp$, if $e \ \to^{lnf^*}\ e'$ then $\llbracket e \rrbracket = \llbracket e' \rrbracket$—and therefore $[\![e]\!] = [\![e']\!]$.

Proof
We first prove the lemma for one step of $\to^{lnf}$ by case distinction over the rule applied to reduce e to e'. By Theorem 9 we already have that $\forall e, e' \in LExp$ if $e \to^{lnf}\ e'$ then $\llbracket e' \rrbracket \Subset \llbracket e \rrbracket$, so all that is left is proving that $\llbracket e \rrbracket \Subset \llbracket e' \rrbracket$ also, and finally applying the transitivity of $\Subset$, as it is a partial order by Lemma 6-i. We proceed assumming some $\theta \in CSubst_\perp$ such that $e\theta \rightarrowtriangle t$ and then proving $e'\theta \rightarrowtriangle t$. The case where $t \equiv\perp$ holds trivially using the rule (B), so we will prove the other by a case distinction on the rule of the *let* calculus applied:

(LetIn) Assume $h(d_1, \ldots, e, \ldots, d_n) \to^l let\ X = e\ in\ h(d_1, \ldots, X, \ldots, d_n)$ by the (LetIn) rule and $\theta \in CSubts_\perp$ such that

$$h(d_1, \ldots, e, \ldots, d_n)\theta \equiv h(d_1\theta, \ldots, e\theta, \ldots, d_n\theta) \rightarrowtriangle t$$

Then by the compositionality of Theorem 5 we have that $\exists t_1 \in \llbracket e\theta \rrbracket$ such that $h(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle t$. Besides X is fresh and $X \notin (dom(\theta) \cup vran(\theta))$ by the variable convention, hence

$$(let\ X = e\ in\ h(d_1, \ldots, X, \ldots, d_n))\theta \equiv let\ X = e\theta\ in\ h(d_1\theta, \ldots, X, \ldots, d_n\theta)$$

and

$$h(d_1\theta, \ldots, X, \ldots, d_n\theta)[X/t_1] \equiv h(d_1\theta, \ldots, t_1, \ldots, d_n\theta)$$

and so we can do:

$$\dfrac{\dfrac{hypothesis}{e\theta \rightarrowtriangle t_1} \quad \dfrac{hypothesis}{h(d_1\theta, \ldots, X, \ldots, d_n\theta)[X/t_1] \equiv h(d_1\theta, \ldots, t_1, \ldots, d_n\theta) \rightarrowtriangle t}}{(let\ X = e\ in\ h(d_1, \ldots, X, \ldots, d_n))\theta \equiv let\ X = e\theta\ in\ h(d_1\theta, \ldots, X, \ldots, d_n\theta) \rightarrowtriangle t}\ (Let)$$

(Bind) Assume $let\ X = t_1\ in\ e \to^l e[X/t_1]$ by (Bind) and $\theta \in CSubst_\perp$ such that $(let\ X = t_1\ in\ e)\theta \equiv let\ X = t_1\theta\ in\ e\theta \rightarrowtriangle t$. Then it must be with a proof of the following shape:

$$\dfrac{t_1\theta \rightarrowtriangle t'_1 \quad e\theta[X/t'_1] \rightarrowtriangle t}{let\ X = t_1\theta\ in\ e\theta \rightarrowtriangle t}\ (Let)$$

But $\theta \in CSubst_\perp$ and $t_1 \in CTerm$ implies $t_1\theta \in CTerm_\perp$, and so $t_1\theta \rightarrowtriangle t'_1$ implies $t'_1 \sqsubseteq t_1\theta$ by Lemma 5-1. Hence $[X/t'_1] \sqsubseteq [X/t_1\theta]$ and so $e\theta[X/t'_1] \rightarrowtriangle t$ implies $e\theta[X/t_1\theta] \rightarrowtriangle t$ by the monoticity of Proposition 5. Besides $X \notin (dom(\theta) \cup vran(\theta))$ by the variable convention, and so we can apply Lemma 1 (substitution lemma) to get $(e[X/t_1])\theta \equiv e\theta[X/t_1\theta]$, so we are done.

(Elim) Assume $let\ X = e_1\ in\ e_2 \to^l e_2$ by (Elim) and $\theta \in CSubts_\perp$ such that $(let\ X = e_1\ in\ e_2)\theta \equiv let\ X = e_1\theta\ in\ e_2\theta \rightarrowtriangle t$. Then it must be with a proof of the following shape:

$$\dfrac{e_1\theta \rightarrowtriangle t_1 \quad e_2\theta[X/t_1] \rightarrowtriangle t}{let\ X = e_1\theta\ in\ e_2\theta \rightarrowtriangle t}\ (Let)$$

Then $X \notin vran(\theta)$ by the variable convention and $X \notin FV(e_2)$ by the condition of (Elim), hence $e_2\theta \equiv e_2\theta[X/t_1] \rightarrowtriangle t$, so we are done.

(Flat) Straightforward since $e_3\theta[Y/t_1] \equiv e_3\theta$ because $Y \notin vran(\theta)$ by the variable convention and $Y \notin FV(e_3)$ by the condition of (Flat).

(Contx) By the proof of the other cases, $\llbracket e \rrbracket \Subset \llbracket e' \rrbracket$, but then $\llbracket \mathcal{C}[e] \rrbracket \Subset \llbracket \mathcal{C}[e'] \rrbracket$ by Lemma 7, and we are done.

□

The following lemmas —Lemmas 29, 30, 31 and 32— will be used to prove Lemma 8.

Lemma 29

Let linear $e, e_1 \in Exp$ such that $e\theta \sqsubseteq e_1$ for $\theta \in Subst_\perp$. Then $\exists \theta' \in Subst$ such that $e\theta' \equiv e_1$ and $\theta \sqsubseteq \theta'$.

Proof

By induction on the structure of e. For the base case ($e \equiv X \in \mathcal{V}$) we define a function $rep_\bot : Exp_\bot \to Exp \to Exp\ rep_\bot(e, e')$ that replaces the occurrences of $\bot$ in e by the expression e'. We define this function recursively on the structure of e:

- $rep_\bot(\bot, e') = e'$
- $rep_\bot(Z, e') = Z$
- $rep_\bot(h(e_1, \ldots, e_n), e') = h(rep_\bot(e_1, e'), \ldots, rep_\bot(e_n, e'))$

It is easy to check that $rep_\bot(e, e') = e''$ implies $e \sqsubseteq e''$. Then we define $\theta' \in Subst$ as:

$$\theta'(Y) = \begin{cases} e_1 & \mathit{if}\ X \equiv Y \\ rep_\bot(\theta(Y), Y) & \mathit{if}\ Y \in dom(\theta) \setminus \{X\} \end{cases}$$

Trivially $e\theta' \equiv X\theta' \equiv e_1$ and $\theta \sqsubseteq \theta'$ because $e\theta \sqsubseteq e_1$ by the premise and $\theta(Y) \sqsubseteq rep_\bot(\theta(Y), Y)$.

Regarding the inductive step —$e \equiv h(e_1, \ldots, e_n)$— we know that

$$e\theta \equiv h(e_1\theta, \ldots, e_n\theta) \sqsubseteq e_1 \equiv h(e'_1, \ldots, e'_n)$$

so $e_i\theta \sqsubseteq e'_i$. Then by IH $\exists \theta'_i \in Subst$ such that $e_i\theta'_i \equiv e'_i$ and $\theta \sqsubseteq \theta'_i$. Then we define θ' as:

$$\theta'(Y) = \begin{cases} \theta'_1(Y) & \mathit{if}\ Y \in var(e_1) \\ \theta'_2(Y) & \mathit{if}\ Y \in var(e_2) \\ \ldots & \\ \theta'_n(Y) & \mathit{if}\ Y \in var(e_n) \\ rep_\bot(\theta(Y), Y) & \mathit{if}\ Y \in dom(\theta) \setminus var(e) \end{cases}$$

The substitution θ' is well defined because e is linear. Then $e\theta' \equiv h(e_1\theta', \ldots, e_n\theta') \equiv h(e_1\theta'_1, \ldots, e_n\theta'_n) = h(e'_1, \ldots, e'_n) \equiv e_1$ and $\theta \sqsubseteq \theta'$ by IH and the fact that $\theta(Y) \sqsubseteq rep_\bot(\theta(Y), Y)$. □

Lemma 30

For any $e \in LExp_\bot$, $FV(|e|) \subseteq FV(e)$.

Proof

Straightforward by induction on the structure of e. □

Lemma 31

Given $e \in LExp$, $\theta \in LSubst_\bot$, $|e\theta| = |e|\hat{\theta}$ where $\hat{\theta}$ is defined as $X\hat{\theta} = |X\theta|$

Proof

By induction on the structure of e. We have two base cases:

- $e \equiv X \in \mathcal{V}$. Then $|e\theta| \equiv |X\theta| = X\hat{\theta} = |X|\theta \equiv |e|\hat{\theta}$.
- $e \equiv f(e_1, \ldots, e_n)$. Then $|e\theta| \equiv |f(e_1, \ldots, e_n)\theta| = |f(e_1\theta, \ldots, e_n\theta)| = \bot = \bot\ \hat{\theta} = |f(e_1, \ldots, e_n)|\hat{\theta} \equiv |e|\hat{\theta}$.

Regarding the inductive step we have:

- $e \equiv c(e_1, \ldots, e_n)$. Straightforward.
- $e \equiv let\ X = e_1\ in\ e_2$. Then $|e\theta| = |(let\ X = e_1\ in\ e_2)\theta| = |let\ X = e_1\theta\ in\ e_2\theta| = |e_2\theta|[X/|e_1\theta|]$. By IH we have that $|e_1\theta| = |e_1|\hat{\theta}$ and $|e_2\theta| = |e_2|\hat{\theta}$, so $|e_2\theta|[X/|e_1\theta|] = |e_2\theta| = (|e_2|\hat{\theta})[X/|e_1|\hat{\theta}]$. By the variable convention we can assume that $X \notin dom(\theta) \cup vran(\theta)$, and since $dom(\hat{\theta}) = dom(\theta)$ and $vran(\hat{\theta}) \subseteq vran(\theta)$ —using Lemma 30— we can use Lemma 1 and obtain $(|e_2|\hat{\theta})[X/|e_1|\hat{\theta}] = (|e_2|[X/|e_1|])\hat{\theta}$. Finally, $(|e_2|[X/|e_1|])\hat{\theta} = |let\ X = e_1\ in\ e_2|\hat{\theta} = |e|\hat{\theta}$.

□

Lemma 32
Given $e \in LExp$, $\theta \in LSubst_\perp$, if $|e| =\perp$ then $|e\theta| =\perp$.

Proof
By induction on the structure of e. Notice that e cannot be a variable X or an applied constructor symbol $c(e_1, \ldots, e_n)$ because in those cases $|e| \neq\perp$. The base case $e \equiv f(e_1, \ldots, e_n)$ is straightforward. Regarding the inductive step we have $e \equiv let\ X = e_1\ in\ e_2$ such that $|let\ X = e_1\ in\ e_2| = |e_2|[X/|e_1|] =\perp$. Then $|e\theta| = |(letX = e_1\ in\ e_2)\theta| = |let\ X = e_1\theta\ in\ e_2\theta| = |e_2\theta|[X/|e_1\theta|]$. By Lemma 23 $|e_2\theta|[X/|e_1\theta|] = |(e_2\theta)[X/e_1\theta]|$, and since $X \notin dom(\theta) \cup vran(\theta)$ by the variable convention then we can apply Lemma 1 and $|(e_2\theta)[X/e_1\theta]| = |(e_2[X/e_1])\theta|$. Finally by Lemma 31 $|(e_2[X/e_1])\theta| = |e_2[X/e_1]|\hat{\theta}$, and by Lemma 23 $|e_2[X/e_1]|\hat{\theta} = (|e_2|[X/|e_1|])\hat{\theta} =\perp \hat{\theta} =\perp$. □

Lemma 8 (Completeness lemma for let-rewriting)
For all $e \in LExp$ and $t \in CTerm_\perp$ such that $t \not\equiv\perp$,

$$e \rightarrowtriangle t \text{ implies } e \to^{l^*} let\ \overline{X = a}\ in\ t'$$

for some $t' \in CTerm$ and $\overline{a} \subseteq LExp$ in such a way that $t \sqsubseteq |let\ \overline{X = a}\ in\ t'|$ and $|a_i| =\perp$ for every $a_i \in \overline{a}$. As a consequence, $t \sqsubseteq t'[\overline{X/\perp}]$.

Proof
By induction on the size s of the $CRWL_{let}$-proof, that we measure as the number of $CRWL_{let}$ rules applied. Concerning the base cases:

(B) This contradicts the hypothesis because then $t \equiv\perp$, so we are done. In the rest of the proof we will assume that $t \not\equiv\perp$ because otherwise we would be in this case.

(RR) Then we have $X \rightarrowtriangle X$. But then $X \to^{l\,0} X$ and $X \sqsubseteq X \equiv |X|$, so we are done with $\overline{X} = \emptyset$.

(DC) Then we have $c \rightarrowtriangle c$. But then $c \to^{l\,0} c$ and $c \sqsubseteq c \equiv |c|$, so we are done with $\overline{X} = \emptyset$.

Now we treat the inductive step:

(DC) Then we have $e \equiv c(e_1, \ldots, e_n)$ and the $CRWL_{let}$-proof has the shape:

$$\frac{e_1 \rightarrowtriangle t_1, \ldots, e_n \rightarrowtriangle t_n}{c(e_1, \ldots, e_n) \rightarrowtriangle c(t_1, \ldots, t_n)}\ (DC)$$

In the general case some t_i will be equal to $\perp$ and some others will be different. For the sake of simplicity we consider the case when $n = 2$ with $t_1 = \perp$ and $t_2 \not\equiv \perp$, the proof can be easily extended to the general case. Then we have $c(e_1, e_2) \mathrel{-\!\!\triangleright} c(\perp, t_2)$, so by IH over the second argument we get $e_2 \to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ t'_2$ with $t'_2 \in \mathit{CTerm}$, $|a_{2_i}| = \perp$ for every $a_{2_i} \in \overline{a_2}$ and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ t'_2| = t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$. So:

$$\begin{array}{ll}
c(e_1, e_2) \to^{l^*} c(e_1, \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ t'_2) & \text{by IH} \\
\to^{l} \mathit{let}\ Y = (\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ t'_2)\ \mathit{in}\ c(e_1, Y) & \text{by (LetIn)} \\
\to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ Y = t'_2\ \mathit{in}\ c(e_1, Y) & \text{by (Flat*)} \\
\to^{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(e_1, t'_2) & \text{by (Bind)}
\end{array}$$

Then there are several possible cases:

a) $e_1 \equiv f_1(\overline{e_1})$: Then $\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(f_1(\overline{e_1}), t'_2) \to^{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ Z = f_1(\overline{e_1})\ \mathit{in}\ c(Z, t'_2)$, by (LetIn). So we are done as $|a_{2_i}| = \perp$ for every a_{2_i} by the IH, $|f_1(\overline{e_1})| = \perp$ and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ Z = f_1(\overline{e_1})\ \mathit{in}\ c(Z, t'_2)| = c(Z, t'_2)[\overline{X_2/\perp}, Z/\perp] \sqsupseteq c(\perp, t_2)$ because $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ by the IH, and Z is fresh and so it does not appear in t'_2

b) $e_1 \equiv t'_1 \in \mathit{CTerm}$: Then we are done as $|a_{2_i}| = \perp$ for every $a_{2_i} \in \overline{a_2}$ by the IH, and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(t'_1, t'_2)| = c(t'_1, t'_2)[\overline{X_2/\perp}] \sqsupseteq c(\perp, t_2)$, because $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ by the IH

c) $e_1 \equiv c_1(\overline{e_1}) \notin \mathit{CTerm}$ with $c_1 \in \mathit{CS}$: Then by Lemma 3 we have the derivation $c_1(\overline{e_1}) \to^{l^*} \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c_1(\overline{t_1})$. But then:

$$\begin{array}{ll}
\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(c_1(\overline{e_1}), t'_2) & \\
\to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(\mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c_1(\overline{t_1}), t'_2) & \text{Lemma 3} \\
\to^{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ Y = (\mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c_1(\overline{t_1}))\ \mathit{in}\ c(Y, t'_2) & \text{by (LetIn)} \\
\to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ \mathit{let}\ Y = c_1(\overline{t_1})\ \mathit{in}\ c(Y, t'_2) & \text{by (Flat*)} \\
\to^{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c(c_1(\overline{t_1}), t'_2) & \text{by (Bind)}
\end{array}$$

In the last step notice that Y is fresh and it cannot appear in t'_2. Then we are done as $|f_i(\overline{t'_i})| = \perp$, $|a_{2_i}| = \perp$ for every $a_{2_i} \in \overline{a_2}$ by the IH, and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c(c_1(\overline{t_1}), t'_2)| = c(c_1(\overline{t_1}), t'_2)[\overline{X_1/\perp}][\overline{X_2/\perp}] \sqsupseteq c(\perp, t_2)$ because $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ by the IH, and no variable in $\overline{X_1}$ appears in t'_2 by α-conversion, as those are bound variables which were present in $c_1(\overline{e_1})$ or that appeared after applying Lemma 3 to it, and this expression was placed in a position parallel to the position of t'_2.

d) $e_1 \equiv \mathit{let}\ X = e_{11}\ \mathit{in}\ e_{12}$: Then by Lemma 3 $\mathit{let}\ X = e_{11}\ \mathit{in}\ e_{12} \to^{l^*} \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ e''$ where $e'' \in \mathcal{V}$ or $e'' \equiv h_1(\overline{t_1})$. Then:

$$\begin{array}{ll}
\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(\mathit{let}\ X = e_{11}\ \mathit{in}\ e_{12}, t'_2) & \\
\to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ c(\mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ e'', t'_2) & \text{by Lemma 3} \\
\to^{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ Y = (\mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ e'')\ \mathit{in}\ c(Y, t'_2) & \text{by (LetIn)} \\
\to^{l^*} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ \mathit{let}\ Y = e''\ \mathit{in}\ c(Y, t'_2) & \text{by (Flat*)}
\end{array}$$

Then we have two possibilities depending on e'':

i) $e'' \equiv Z \in \mathcal{V}$: Then we can do:

$$\begin{array}{ll} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ \mathit{let}\ Y = Z\ \mathit{in}\ c(Y, t'_2) & \\ \to^l \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c(Z, t'_2) & \text{by (Bind)} \end{array}$$

Then we are done as $|f_1(\overline{t'_1})| =\perp$, $|a_{2_i}| =\perp$ for every $a_{2_i} \in \overline{a_2}$ by IH, and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c(Z, t'_2)| = c(Z, t'_2)[\overline{X_1/\perp}][\overline{X_2/\perp}] \sqsupseteq c(\perp, t_2)$, as $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ by IH, and no variable in $\overline{X_1}$ appears in t'_2 by α-conversion, like in the case *c)*.

ii) $e'' \equiv h_1(\overline{t_1})$: there are two possible cases:

A) $h_1 = f_1 \in FS$: We are done as $|f_1(\overline{t'_1})| =\perp$, $|a_{2_i}| =\perp$ for every $a_{2_i} \in \overline{a_2}$ by IH, $|f_1(\overline{t_1})| =\perp$, and $|\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ \mathit{let}\ Y = f_1(\overline{t_1})\ \mathit{in}\ c(Y, t'_2)| = c(Y, t'_2)[Y/\perp][\overline{X_1/\perp}][\overline{X_2/\perp}] \sqsupseteq c(\perp, t_2)$, as by IH $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$, Y is fresh and so it does not appear in t'_2, and no variable in $\overline{X_1}$ appears in t'_2 as in the case *i)*.

B) $h_1 = c_1 \in DC$: Then we can do a (Bind) step:

$$\begin{array}{l} \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ \mathit{let}\ Y = c_1(\overline{t_1})\ \mathit{in}\ c(Y, t'_2) \\ \to^l \mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ c(c_1(\overline{t_1}), t'_2) \end{array}$$

Then we are done as $|f_1(\overline{t'_1})| =\perp$, $|a_{2_i}| =\perp$ for every $a_{2_i} \in \overline{a_2}$ by IH, and

$$\begin{array}{rl} & |\mathit{let}\ \overline{X_2 = a_2}\ \mathit{in}\ \mathit{let}\ \overline{X_1 = f_1(\overline{t'_1})}\ \mathit{in}\ c(c_1(\overline{t_1}), t'_2)| \\ = & c(c_1(\overline{t_1}), t'_2)[\overline{X_1/\perp}][\overline{X_2/\perp}] \\ \sqsupseteq & c(\perp, t_2) \end{array}$$

as $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ by IH, and no variable in $\overline{X_1}$ appears in t'_2, as we saw in *i)*.

(OR) If f has no arguments ($n = 0$) then we have:

$$\frac{r\theta \rightarrowtriangle t}{f \rightarrowtriangle t}\ (OR)$$

with $(f \rightarrowtriangle r) \in \mathcal{P}$ and $\theta \in CSubst_\perp$. Let us define $\theta' \in CSubst$ as the substitution which is equal to θ except that every $\perp$ introduced by θ is replaced with some constructor symbol or variable. Then $\theta \sqsubseteq \theta'$, so by Proposition 5 we have $r\theta' \rightarrowtriangle t$ with a proof of the same size. But then applying the IH to this proof we get $r\theta' \to^{l^*} \mathit{let}\ \overline{X = a}\ \mathit{in}\ t'$ under the conditions of the lemma. Hence $f \to^l r\theta' \to^{l^*} \mathit{let}\ \overline{X = a}\ \mathit{in}\ t'$ applying (Fapp) in the first step, and we are done.

If $n > 0$, we will proceed as in the case for (DC), doing a preliminary version for $f(e_1, e_2) \rightarrowtriangle t$ which can be easily extended for the general case. Then we have:

$$\frac{e_1 \rightarrowtriangle \perp \quad e_2 \rightarrowtriangle t_2 \quad r\theta \rightarrowtriangle t}{f(e_1, e_2) \rightarrowtriangle t}\ (OR)$$

such that $t_2 \not\equiv \perp$, and with $(f(p_1, p_2) \to r) \in \mathcal{P}$, $\theta \in CSubst_\perp$, such that

$p_1\theta \equiv\perp$ and $p_2\theta = t_2$. Then applying the IH to $e_2 \rightarrowtriangle t_2$ we get that $e_2 \to^{l^*}$ $let\ \overline{X_2 = a_2}\ in\ t'_2$ such that $|a_{2_i}| =\perp$ for every a_{2_i} and $|let\ \overline{X_2 = a_2}\ in\ t'_2| = t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$. Then we can do:

$$\begin{array}{ll}
f(e_1, e_2) \to^{l^*} f(e_1, let\ \overline{X_2 = a_2}\ in\ t'_2) & \text{by IH} \\
\to^{l} let\ Y = (let\ \overline{X_2 = a_2}\ in\ t'_2)\ in\ f(e_1, Y) & \text{by (LetIn)} \\
\to^{l^*} let\ \overline{X_2 = a_2}\ in\ let\ Y = t'_2\ in\ f(e_1, Y) & \text{by (Flat*)} \\
\to^{l} let\ \overline{X_2 = a_2}\ in\ f(e_1, t'_2) & \text{by (Bind)}
\end{array}$$

Then applying Lemma 3 we get

$$f(e_1, t'_2) \to^{l^*} let\ \overline{X_1 = f_1(\overline{t'})}\ in\ f(t'_1, t'_2)$$

Now as $t'_2[\overline{X_2/\perp}] \sqsupseteq t_2$ then $(t'_1, t'_2) \sqsupseteq (\perp, t_2)$, so by Lemma 29 there must exist $\theta' \in CSubst$ such that $\theta \sqsubseteq \theta'$ and $(p_1, p_2)\theta' \equiv (t'_1, t'_2)$. Then by Proposition 5, as $r\theta \rightarrowtriangle t$ then $r\theta' \rightarrowtriangle t$ with a proof of the same size. As $\theta' \in CSubst$ and $e \in LExp$ (because it is part of the program) then $r\theta' \in LExp$ and we can apply the IH to that proof getting that $r\theta' \to^{l^*} let\ \overline{X = a}\ in\ t'$ such that $|a_i| =\perp$ for every a_i and $|let\ \overline{X = a}\ in\ t'| = t'[\overline{X/\perp}] \sqsupseteq t$. Then we can do:

$$\begin{array}{ll}
let\ \overline{X_2 = a_2}\ in\ f(e_1, t'_2) & \\
\to^{l^*} let\ \overline{X_2 = a_2}\ in\ let\ \overline{X_1 = f_1(\overline{t'})}\ in\ f(t'_1, t'_2) & \text{by Lemma 3} \\
\equiv let\ \overline{X_2 = a_2}\ in\ let\ \overline{X_1 = f_1(\overline{t'})}\ in\ f(p_1, p_2)\theta' & \\
\to^{l} let\ \overline{X_2 = a_2}\ in\ let\ \overline{X_1 = f_1(\overline{t'})}\ in\ r\theta' & \text{by (Fapp)} \\
\to^{l^*} let\ \overline{X_2 = a_2}\ in\ let\ \overline{X_1 = f_1(\overline{t'})}\ in\ let\ \overline{X = a}\ in\ t' & \text{by } 2^{nd} \text{ IH}
\end{array}$$

Then $|a_{2_i}| =\perp$ for every $a_{2_i} \in \overline{a_2}$ by IH, $|f_1(\overline{t'})| =\perp$ and $|a_i| =\perp$ for every a_i by IH. Besides the variables in $\overline{X_1} \cup \overline{X_2}$ either belong to $BV(e_1) \cup BV(e_2)$ or are fresh, hence none of them may appear in t (by Lemma 27 over $f(e_1, e_2) \rightarrowtriangle t$ or by freshness). So $t'[\overline{X/\perp}] \sqsupseteq t$ implies that $\forall p \in O(t')$ such that $t'|_p = Y$ for some $Y \in \overline{X_1} \cup \overline{X_2}$ then $t|_p =\perp$. But then $|let\ \overline{X_2 = a_2}\ in\ let\ \overline{X_1 = a_1}\ in\ let\ \overline{X = a}\ in\ t'| \equiv t'[\overline{X/\perp}][\overline{X_1/\perp}][\overline{X_2/\perp}] \sqsupseteq t$.

(Let) Then $e \equiv let\ X = e_1\ in\ e_2$ and we have a proof of the following shape:

$$\frac{e_1 \rightarrowtriangle t_1 \quad e_2[X/t_1] \rightarrowtriangle t}{let\ X = e_1\ in\ e_2 \rightarrowtriangle t}\ (Let)$$

Then we have two possibilities:

a) $t_1 \equiv\perp$: Then $e_2[X/t_1] \equiv e_2[X/\perp] \sqsubseteq e_2$. Hence, as $e_2[X/t_1] \rightarrowtriangle t$ and $[X/t_1] \sqsubseteq \epsilon$, by Proposition 5 we get $e_2\epsilon \equiv e_2 \rightarrowtriangle t$ with a proof of the same size or smaller, and so by IH we get $e_2 \to^{l^*} let\ \overline{X = a}\ in\ t'$, with $t' \in CTerm$, $|a_i| \equiv\perp$ for every a_i and $|let\ \overline{X = a}\ in\ t'| \equiv t'[\overline{X/\perp}] \sqsupseteq t$, and we can do:

$$let\ X = e_1\ in\ e_2 \to^{l^*} let\ X = e_1\ in\ let\ \overline{X = a}\ in\ t'$$

Besides $X \not\in var(t)$ by Lemma 27 over $let\ X = e_1\ in\ e_2 \rightarrowtriangle t$, and then $t'[\overline{X/\perp}] \sqsupseteq t$ implies $\forall p \in O(t')$ such that $t'|_p \equiv X$ then $t|_p \equiv\perp$, and we have several possible cases:

i) $e_1 = f_1(\overline{e_1})$: Then we are donde because $|\overline{a}| \equiv \overline{\perp}$ by IH, $|f_1(\overline{e_1})| \equiv\perp$ and $|let\ X = f_1(\overline{e_1})\ in\ let\ \overline{X = a}\ in\ t'| \equiv t'[\overline{X/\perp}][X/\perp] \sqsupseteq t$, as $t'[\overline{X/\perp}] \sqsupseteq t$ and $\forall p \in O(t')$ such that $t'|_p \equiv X$ then $t|_p \equiv\perp$, as we saw above.

ii) $e_1 = t'_1 \in CTerm$: But then

$$let\ X = t'_1\ in\ let\ \overline{X = a}\ in\ t' \to^l let\ \overline{X = a[X/t'_1]}\ in\ t'[X/t'_1] \quad \text{by (Bind)}$$

and we are done because $|\overline{a}| \equiv \overline{\perp}$ by IH, and so $|\overline{a}[X/t'_1]| \equiv \overline{\perp}$ by Lemma 32. Besides, as in *i)*, $t'[\overline{X/\perp}] \sqsupseteq t$ combined with the fact that $\forall p \in O(t')$ such that $t'|_p \equiv X$ we have $t|_p \equiv\perp$, implies that $|let\ \overline{X = a[X/t'_1]}\ in\ t'[X/t'_1]| \equiv t'[X/t'_1][\overline{X/\perp}] \sqsupseteq t$.

iii) $e_1 = c_1(\overline{e_1}) \notin CTerm$ with $c_1 \in CS$: Then by Lemma 3 we have $c_1(\overline{e_1}) \to^{l^*} let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ c_1(\overline{t_1})$, hence

$$\begin{array}{ll} let\ X = c_1(\overline{e_1})\ in\ let\ \overline{X = a}\ in\ t' & \\ \to^{l^*} let\ X = (let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ c_1(\overline{t_1}))\ in\ let\ \overline{X = a}\ in\ t' & \text{by Lemma 3} \\ \to^{l^*} let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ let\ X = c_1(\overline{t_1})\ in\ let\ \overline{X = a}\ in\ t' & \text{by (Flat}^*\text{)} \\ \to^{l} let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ let\ \overline{X = a[X/c_1(\overline{t_1})]}\ in\ t'[X/c_1(\overline{t_1})] & \text{by (Bind)} \end{array}$$

As by IH $|\overline{a}| \equiv \overline{\perp}$ then $|\overline{a[X/c_1(\overline{t_1})]}| \equiv \overline{\perp}$ by Lemma 32. At this point we have to check that $|let\ \overline{X_1 = a_1}\ in\ let\ \overline{X = a[X/c_1(\overline{t_1})]}\ in\ t'[X/c_1(\overline{t_1})]| \equiv t'[X/c_1(\overline{t_1})][\overline{X/\perp}][\overline{X_1/\perp}] \sqsupseteq t$. The variables in $\overline{X_1}$ either belong to $BV(c_1(\overline{e_1}))$ or are fresh, hence by α-conversion none of them may appear in t', because in $let\ X = c_1(\overline{e_1})\ in\ let\ \overline{X = a}\ in\ t'$ the expression t' has no access to the variables bound in $c_1(\overline{e_1})$. Hence $t'[X/c_1(\overline{t_1})][\overline{X/\perp}][\overline{X_1/\perp}] \equiv t'[X/t''][\overline{X/\perp}]$, for some $t'' \in CTerm_\perp$. But then, as in *ii)*, $t'[\overline{X/\perp}] \sqsupseteq t$ combined with the fact that $\forall p \in O(t')$ such that $t'|_p \equiv X$ we have $t|_p \equiv\perp$, implies that $t'[X/t''][\overline{X/\perp}] \sqsupseteq t$.

iv) $e_1 \equiv let\ Y = e_{11}\ in\ e_{12}$: Then by Lemma 3 we have $let\ Y = e_{11}\ in\ e_{12} \to^{l^*} let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ h_1(\overline{t_1})$, and so

$$\begin{array}{ll} let\ X = (let\ Y = e_{11}\ in\ e_{12})\ in\ let\ \overline{X = a}\ in\ t' & \\ \to^{l^*} let\ X = (let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ h_1(\overline{t_1}))\ in\ let\ \overline{X = a}\ in\ t' & \text{by Lemma 3} \\ \to^{l^*} let\ \overline{X_1 = f_1(\overline{t_1})}\ in\ let\ X = h_1(\overline{t_1})\ in\ let\ \overline{X = a}\ in\ t' & \text{by (Flat}^*\text{)} \end{array}$$

Then either $h \in CS$ and we are like in *iii)* before the final (Bind) step, or $h \in FS$ and $|h_1(\overline{t_1})| =\perp$ and $|\overline{a}| = \overline{\perp}$ (by IH), and $|let\ \overline{X_1 = a_1}\ in\ let\ X = h_1(\overline{t_1})\ in\ let\ \overline{X = a}\ in\ t'| \equiv t'[\overline{X/\perp}][X/\perp][\overline{X_1/\perp}] \equiv t'[\overline{X/\perp}][X/\perp]$ because $\overline{X_1} \cap var(t') = \emptyset$, as we saw in *iii)*. But then, as in *ii)*, $t'[\overline{X/\perp}] \sqsupseteq t$ combined with the fact that $\forall p \in O(t')$ such that $t'|_p \equiv X$ we have $t|_p \equiv\perp$, implies that $t'[\overline{X/\perp}][X/\perp] \sqsupseteq t$.

b) $t_1 \not\equiv\perp$: Then by IH we get $e_1 \to^{l^*} let\ \overline{X_1 = a_1}\ in\ t'_1$, with $t'_1 \in CTerm$, $|a_{1_i}| \equiv\perp$ for every a_{1_i} and $|let\ \overline{X_1 = a_1}\ in\ t'_1| \equiv t'_1[\overline{X_1/\perp}] \sqsupseteq t_1$. Hence $t_1 \sqsubseteq t'_1$ and so $e_2[X/t_1] \sqsubseteq e_2[X/t'_1]$, but then $e_2[X/t_1] \rightarrowtriangle t$ implies $e_2[X/t'_1] \rightarrowtriangle t$ with a proof of the same size or smaller, by Proposition 3. Therefore we may apply the IH to that proof to get $e_2[X/t'_1] \to^{l^*} let\ \overline{X = a}\ in\ t'$, with $t' \in CTerm$, $|a_i| \equiv\perp$ for every a_i and $|let\ \overline{X = a}\ in\ t'| \equiv t'[\overline{X/\perp}] \sqsupseteq t$. But then we can do:

$$\begin{array}{ll}
\mathit{let}\ X = e_1\ \mathit{in}\ e_2 \rightarrow^{l^*} \mathit{let}\ X = (\mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ t'_1)\ \mathit{in}\ e_2 & \text{by IH} \\
\rightarrow^{l^*} \mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ \mathit{let}\ X = t'_1\ \mathit{in}\ e_2 & \text{by (Flat}^*\text{)} \\
\rightarrow^{l} \mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ e_2[X/t'_1] & \text{by (Bind)} \\
\rightarrow^{l^*} \mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ \mathit{let}\ \overline{X = a}\ \mathit{in}\ t' & \text{by IH}
\end{array}$$

Then by the IH's we have $|\overline{a}| = \overline{\perp}$ and $|\overline{a_1}| = \overline{\perp}$. Besides the variables in $\overline{X_1}$ either belong to $BV(e_1)$ or are fresh, hence none of them may appear in t (by Lemma 27 over $\mathit{let}\ X = e_1\ \mathit{in}\ e_2 \twoheadrightarrow t$ or by freshness). So $t'[\overline{X/\perp}] \sqsupseteq t$ implies that $\forall p \in O(t')$ such that $t'|_p = Y$ for some $Y \in \overline{X_1}$ then $t|_p = \perp$. But then $|\mathit{let}\ \overline{X_1 = a_1}\ \mathit{in}\ \mathit{let}\ \overline{X = a}\ \mathit{in}\ t'| \equiv t'[\overline{X/\perp}][\overline{X_1/\perp}] \sqsupseteq t$.

□

A.8 Proofs for Section 5

Lemma 10

If $BV(\mathcal{C}) \cap FV(e_1) = \emptyset$ and $X \notin FV(\mathcal{C})$ then $\llbracket \mathcal{C}[\mathit{let}\ X = e_1\ \mathit{in}\ e_2] \rrbracket = \llbracket \mathit{let}\ X = e_1\ \mathit{in}\ \mathcal{C}[e_2] \rrbracket$

Proof

One step of the rule (Dist) can be replaced by two steps (CLetIn) + (Bind):

$$\mathcal{C}[\mathit{let}\ X = e_1\ \mathit{in}\ e_2] \rightarrow^{l} \mathit{let}\ U = e_1\ \mathit{in}\ \mathcal{C}[\mathit{let}\ X = U\ \mathit{in}\ e_2] \rightarrow^{l} \mathit{let}\ U = e_1\ \mathit{in}\ \mathcal{C}[e_2[X/U]]$$

followed by a renaming of U by X in the last expression. Then the lemma follows from preservation of hypersemantics by (CLetIn) and (Bind) (Lemma 9 and Proposition 8). □

Proposition 9 ((Hyper)semantic properties of ?)

For any $e_1, e_2 \in LExp_{\perp}$

i) $\llbracket e_1\ ?\ e_2 \rrbracket = \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket$
ii) $\llbracket\!\llbracket e_1\ ?\ e_2 \rrbracket\!\rrbracket = \llbracket\!\llbracket e_1 \rrbracket\!\rrbracket \Cup \llbracket\!\llbracket e_2 \rrbracket\!\rrbracket$

Proof

i) Direct from definition of ? and the CRWL-proof calculus.
ii)

$$\begin{array}{ll}
\llbracket\!\llbracket e_1\ ?\ e_2 \rrbracket\!\rrbracket = \lambda\theta.\llbracket (e_1\ ?\ e_2)\theta \rrbracket & \text{by definition of } \llbracket\!\llbracket\ \rrbracket\!\rrbracket \\
= \lambda\theta.\llbracket e_1\theta\ ?\ e_2\theta \rrbracket & \\
= \lambda\theta.(\llbracket e_1\theta \rrbracket\ \cup\ \llbracket e_2\theta \rrbracket) & \text{by } \mathit{i)} \\
= \lambda\theta.(\llbracket\!\llbracket e_1 \rrbracket\!\rrbracket\theta\ \cup\ \llbracket\!\llbracket e_2 \rrbracket\!\rrbracket\theta) & \text{by definition of } \llbracket\!\llbracket\ \rrbracket\!\rrbracket \\
= \llbracket\!\llbracket e_1 \rrbracket\!\rrbracket \Cup \llbracket\!\llbracket e_2 \rrbracket\!\rrbracket & \text{by definition of } \Cup
\end{array}$$

□

A.9 Proofs for Section 6

Theorem 14 (Soundness of the let-narrowing relation $\leadsto^{l}$)

For any $e, e' \in LExp$, $e \leadsto^{l^*}_{\theta} e'$ implies $e\theta \rightarrow^{l}{}^{*} e'$.

Proof
First we prove the soundness of narrowing for one step, proceeding by a case distinction over the rule used in $e \rightsquigarrow^{l}{}_{\theta}\ e'$. The cases of (Elim), (Bind), (Flat) and (LetIn) are trivial, since narrowing and rewriting coincide for these rules.

(Narr) Then we have $f(\bar{t}) \rightsquigarrow^{l}{}_{\theta}\ r\theta$ for $(f(\overline{p}) \to r) \in \mathcal{P}$ fresh, $\theta \in CSubst$ such that $f(\bar{t})\theta \equiv f(\overline{p})\theta$. But then $(f(\overline{p}) \to r)\theta \equiv f(\overline{p})\theta \to r\theta \equiv f(\bar{t})\theta \to r\theta$, so we can do $e\theta \equiv f(\bar{t})\theta \to^{l} r\theta \equiv e'$ by (Fapp).

(Contxt) Then we have $\mathcal{C}[e] \rightsquigarrow^{l}{}_{\theta}\ \mathcal{C}\theta[e']$ because $e \rightsquigarrow^{l}{}_{\theta} e'$. Let us do a case distinction over the rule applied in $e \rightsquigarrow^{l}{}_{\theta}\ e'$:

a) $e \rightsquigarrow^{l}{}_{\theta}\ e' \equiv f(\bar{t}) \rightsquigarrow^{l}{}_{\theta}\ r\theta$ by (Narr), for $(f(\overline{p}) \to r) \in \mathcal{P}$ fresh, so $f(\bar{t})\theta \to^{l} r\theta$ by (Fapp). Then $(\mathcal{C}[e])\theta \equiv (\mathcal{C}[e])\theta|_{\backslash var(\overline{p})}$, because the variables in $var(\overline{p})$ are fresh as $(f(\overline{p}) \to r)$ is. But then, as $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ and $vRan(\theta|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$ by the conditions in (Contx), and $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ implies $dom(\theta|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$, we can apply Lemma 25 getting $(\mathcal{C}[e])\theta|_{\backslash var(\overline{p})} \equiv \mathcal{C}\theta|_{\backslash var(\overline{p})}[e\theta|_{\backslash var(\overline{p})}] \equiv \mathcal{C}\theta|_{\backslash var(\overline{p})}[f(\bar{t})\theta|_{\backslash var(\overline{p})}] \equiv \mathcal{C}\theta[f(\bar{t})\theta]$, because the variables in $var(\overline{p})$ are fresh. Besides $vran(\theta|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$, so we can apply (Contx) combined with an inner (Fapp) to do $(\mathcal{C}[e])\theta \equiv \mathcal{C}\theta[f(\bar{t})\theta] \to^{l} \mathcal{C}\theta[r\theta] \equiv \mathcal{C}\theta[e']$.

b) In case a different rule was applied in $e \rightsquigarrow^{l}{}_{\theta}\ e'$ then $\theta = \epsilon$. By the proof of the other cases we have $e\theta \equiv e \to^{l} e'$, so $(\mathcal{C}[e])\theta \equiv \mathcal{C}[e] \to^{l} \mathcal{C}[e'] \equiv \mathcal{C}\theta[e']$ (remember $\theta = \epsilon$).

Now we prove the lemma for any number of steps $\to^{l}$, proceeding by induction over the length n of $e \rightsquigarrow^{l^n}_{\theta}\ e'$. The case $e \rightsquigarrow^{l^0}_{\epsilon}\ e \equiv e'$ is straightforward because $e \to^{l^0}\ e \equiv e'$. For $n > 0$ we have the derivation $e \rightsquigarrow^{l}_{\sigma}\ e'' \rightsquigarrow^{l^{n-1}}_{\gamma}\ e'$ with $\theta = \gamma \circ \sigma$. By the proof for one step $e\sigma \to^{l} e''$, and by the closeness under $CSubst$ of let-rewriting (Lemma 2) $e\sigma\gamma \to^{l} e''\gamma$. By IH $e''\gamma \to^{l^*}\ e'$, so we can link $e\theta \equiv e\sigma\gamma \to^{l} e''\gamma \to^{l^*}\ e'$.
□

Lemma 11 (Lifting lemma for the let-rewriting relation $\to^{l}$)
Let $e, e' \in LExp$ such that $e\theta \to^{l^*}\ e'$ for some $\theta \in CSubst$, and let $\mathcal{W}, \mathcal{B} \subseteq \mathcal{V}$ with $dom(\theta) \cup FV(e) \subseteq \mathcal{W}$, $BV(e) \subseteq \mathcal{B}$ and $(dom(\theta) \cup vran(\theta)) \cap \mathcal{B} = \emptyset$, and for each (Fapp) step of $e\theta \to^{l^*}\ e'$ using a rule $R \in \mathcal{P}$ and a substitution $\gamma \in CSubst$ then $vran(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$. Then there exist a derivation $e \rightsquigarrow^{l^*}_{\sigma}\ e''$ and $\theta' \in CSubst$ such that:

(i) $e''\theta' = e'$ (ii) $\sigma\theta' = \theta[\mathcal{W}]$ (iii) $(dom(\theta') \cup vran(\theta')) \cap \mathcal{B} = \emptyset$

Besides, the let-narrowing derivation can be chosen to use mgu's at each (Narr) step.

Proof
Let us do a case distinction over the rule applied in $e\theta \to^{l} e'$:

(Fapp) $e \equiv f(\overline{t})$, so:

$$\begin{tikzcd}
e \equiv f(\overline{t}) \arrow[r, squiggly, dotted, "l", "\sigma"'] \arrow[d, mapsto, "\theta"'] & r\sigma \equiv e'' \arrow[d, mapsto, dashed, "\theta'"] \\
f(\overline{t})\theta \arrow[r, "l"'] & r\gamma \equiv e'
\end{tikzcd}$$

With an (Fapp) step $e\theta \equiv f(\overline{t})\theta \rightarrow^{l} r\gamma$ with $(f(\overline{p}) \rightarrow r) \in \mathcal{P}$, $\gamma \in CSubst$, such that $f(\overline{t})\theta \equiv f(\overline{p})\gamma$ and $f(\overline{p}) \rightarrow r$ is a fresh variant. We can assume that $dom(\gamma) \subseteq FV(f(\overline{p}) \rightarrow r)$ without loss of generality. But then $dom(\theta) \cap dom(\gamma) = \emptyset$, and so $\theta \uplus \gamma$ is correctly defined, and it is a unifier of $f(\overline{t})$ and $f(\overline{p})$. So, there must exist $\sigma = mgu(f(\overline{t}), f(\overline{p}))$, which we can use to perform a (Narr) step, because $\sigma \in CSubst$ and $f(\overline{t})\sigma \equiv f(\overline{p})\sigma$.

$$e \equiv f(\overline{t}) \rightsquigarrow^{l}{}_{\sigma}\ r\sigma \equiv e''$$

As this unifier is an mgu then $dom(\sigma) \subseteq FV(f(\overline{t})) \cup FV(f(\overline{p}))$, $vran(\sigma) \subseteq FV(f(\overline{t})) \cup FV(f(\overline{p}))$ and $\sigma \lesssim (\theta \uplus \gamma)$, so there must exist $\theta'_1 \in CSubst$ such that $\sigma\theta'_1 = \theta \uplus \gamma$. Besides we can define $\theta'_0 = \theta|_{\setminus(dom(\theta'_1) \cup FV(f(\overline{t})))}$ and then we can take $\theta' = \theta'_0 \uplus \theta'_1$ which is correctly defined as obviously $dom(\theta'_0) \cap dom(\theta'_1) = \emptyset$. Besides $dom(\theta'_0) \cap (FV(f(\overline{t})) \cup FV(f(\overline{p})) = \emptyset$, as if $Y \in FV(f(\overline{t}))$ then $Y \notin dom(\theta'_0)$ by definition; and if $Y \in FV(f(\overline{p}))$ then $Y \notin dom(\theta)$ as $\overline{p}$ belong to the fresh variant, and so $Y \notin dom(\theta'_0)$. Then the conditions in Lemma 11 hold:

- Condition i) $e''\theta' \equiv e'$: As $e''\theta' \equiv r\sigma\theta' \equiv r\sigma\theta'_1$ because given $Y \in FV(r\sigma)$, if $Y \in FV(r)$ then it belongs to the fresh variant and so $Y \notin dom(\theta) \supseteq dom(\theta'_0)$; and if $Y \in vran(\sigma) \subseteq FV(f(\overline{t})) \cup FV(f(\overline{p}))$ then $Y \notin dom(\theta'_0)$ because $dom(\theta'_0) \cap (FV(f(\overline{t})) \cup FV(f(\overline{p}))) = \emptyset$. But $r\sigma\theta'_1 \equiv r(\theta \uplus \gamma) \equiv r\gamma \equiv e'$, because $\sigma\theta'_1 = \theta \uplus \gamma$ and r is part of the fresh variant.
- Condition ii) $\sigma\theta' = \theta[\mathcal{W}]$: Given $Y \in \mathcal{W}$, if $Y \in FV(f(\overline{t}))$ then $Y \notin dom(\gamma)$ and so $Y\theta \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$, as $\sigma\theta'_1 = \theta \uplus \gamma$. But $Y\sigma\theta'_1 \equiv Y\sigma\theta'$ because given $Z \in var(Y\sigma)$, if $Z \equiv Y$ then as $Y \in FV(f(\overline{t}))$ then $Z \equiv Y \notin dom(\theta'_0)$ by definition of θ'_0; if $Z \in vran(\sigma)$ then $Z \notin dom(\theta'_0)$, as we saw before. On the other hand, $(\mathcal{W} \setminus FV(f(\overline{t}))) \cap (FV(f(\overline{t})) \cup FV(f(\overline{p}))) = (\mathcal{W} \setminus FV(f(\overline{t})) \cap FV(f(\overline{t}))) \cup (\mathcal{W} \setminus FV(f(\overline{t})) \cap FV(f(\overline{p}))) = \emptyset \cup \emptyset = \emptyset$, because $FV(f(\overline{p}))$ are part of the fresh variant. So, if $Y \in \mathcal{W} \setminus FV(f(\overline{t}))$, then $Y \notin dom(\sigma) \subseteq FV(f(\overline{t})) \cup FV(f(\overline{p}))$. Now if $Y \in dom(\theta'_0)$ then $Y\theta \equiv Y\theta'_0$ (by definition of θ'_0), $Y\theta'_0 \equiv Y\theta'$ (as $Y \in dom(\theta'_0)$), $Y\theta' \equiv Y\sigma\theta'$ (as $Y \notin dom(\sigma)$). If $Y \in dom(\theta'_1)$, $Y\theta \equiv Y(\theta \uplus \gamma)$ (as $Y \in \mathcal{W} \setminus FV(f(\overline{t}))$ implies it does not appear in the fresh instance), $Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$ (as $\sigma\theta'_1 = \theta \uplus \gamma$), $Y\sigma\theta'_1 \equiv Y\theta'_1$ (as $Y \notin dom(\sigma)$), $Y\theta'_1 \equiv Y\theta'$ (as $Y \in dom(\theta'_1)$) and $Y\theta' \equiv Y\sigma\theta'$ (as $Y \notin dom(\sigma)$). And if $Y \notin (dom(\theta'_0) \cup dom(\theta'_1))$ then $Y \notin dom(\theta')$, and as $Y \notin dom(\sigma)$ and $Y\theta \equiv Y(\theta \uplus \gamma)$, then $Y\theta \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1 \equiv Y \equiv Y\sigma\theta'$.

- Condition iii.1) $dom(\theta') \cap \mathcal{B} = \emptyset$. Remember $\theta' = \theta'_0 \uplus \theta'_1$:

— $dom(\theta'_0) \cap \mathcal{B} = \emptyset$: Given $Y \in dom(\theta'_0)$ then $Y \in dom(\theta)$ by definition of θ'_0, and so $Y \notin \mathcal{B}$, because $dom(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis.

— $dom(\theta'_1) \cap \mathcal{B} = \emptyset$: As σ is an mgu and $\sigma \lesssim \theta \uplus \gamma$, then $dom(\sigma) \subseteq dom(\theta \uplus \gamma)$. Given $Z \in \mathcal{B}$ then $Z \notin dom(\theta)$, as $dom(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis, and $Z \notin dom(\gamma) \subseteq FV(f(\overline{p}) \to r)$ which are fresh, so $Z \notin dom(\sigma)$. But then, as $\sigma\theta'_1 = \theta \uplus \gamma$, $Z \equiv Z(\theta \uplus \gamma) \equiv Z\sigma\theta'_1 \equiv Z\theta'_1$, so $Z \notin dom(\theta'_1)$.

- Condition iii.2) $vran(\theta') \cap \mathcal{B} = \emptyset$. Remember $\theta' = \theta'_0 \uplus \theta'_1$:

— $vran(\theta'_0) \cap \mathcal{B} = \emptyset$: Given $Y \in dom(\theta'_0)$ then $Y\theta'_0 \equiv Y\theta$ by definition of θ'_0. As $vran(\theta) \cap \mathcal{B} = \emptyset$ by hypothesis then it must happen $var(Y\theta) \cap \mathcal{B} = \emptyset$, so $var(Y\theta'_0) \cap \mathcal{B} = \emptyset$.

— $vran(\theta'_1) \cap \mathcal{B} = \emptyset$: As $\sigma\theta'_1 = \theta \uplus \gamma$ then we can assume $dom(\theta'_1) \subseteq vran(\sigma) \cup (dom(\theta \uplus \gamma) \setminus dom(\sigma))$.

  – Let $X \in dom(\theta'_1) \cap vran(\sigma)$ be such that $X\theta'_1 \equiv r[Z]$ with $Z \in \mathcal{B}$. We will see that this $Z \in \mathcal{B}$ can appear in $X\theta'_1$ without leading to contradiction. The intuition is, as $vran(\theta) \cap \mathcal{B} = \emptyset$ and $vran(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$, then every $Z \in \mathcal{B}$ must come from an appearance in e of the same variable, transmitted to e' by the matching substitution γ, and so transmitted to e'' by σ.

  As $X \in vran(\sigma)$ then there must exist $Y \in dom(\sigma)$ such that $Y \longmapsto^{\sigma} r_1[X]_p \longmapsto^{\theta'_1} r_2[s[Z]]_p$. But as $\sigma\theta'_1 = \theta \uplus \gamma$ then $Y \longmapsto^{\theta \uplus \gamma} r_2[s[Z]]_p$. Then, $Z \in vran(\theta \uplus \gamma)$, but $Z \in \mathcal{B}$, $vran(\theta) \cap \mathcal{B} = \emptyset$, $vran(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$, $dom(\gamma) \subseteq FV(f(\overline{p}) \to s)$, so it must happen $Z \in vran(\gamma|_{FV(\overline{p})})$, and as a consequence $Y \in FV(\overline{p})$. Let $o \in O(f(\overline{p}))$ (set of positions in $f(\overline{p})$) be such that $f(\overline{p})|_o \equiv Y$, then:

  · $((f(\overline{t}))\sigma)|_o \equiv ((f(\overline{p}))\sigma)|_o \equiv ((f(\overline{p}))|_o)\sigma \equiv Y\sigma \equiv r_1[X]_p$.

  · As $f(\overline{t}) \notin dom(\gamma)$, which are the fresh variables of the variant of the program rule, $((f(\overline{t}))\theta)|_o \equiv ((f(\overline{t}))(\theta \uplus \gamma))|_o \equiv ((f(\overline{p}))(\theta \uplus \gamma))|_o \equiv ((f(\overline{p}))|_o)(\theta \uplus \gamma) \equiv Y(\theta \uplus \gamma) \equiv r_2[s[Z]]_p$

  So, as $X \in dom(\theta'_1)$ then $X \notin \mathcal{B}$ and $Z \in \mathcal{B}$ has been introduced by θ, but this is impossible as $vran(\theta) \cap \mathcal{B} = \emptyset$.

  – Let $Y \in dom(\theta) \setminus dom(\sigma)$ be. Then $Y\theta \equiv Y(\theta \uplus \gamma)$ (as $Y \in dom(\theta)$, $Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1$ (as $\sigma\theta'_1 = \theta \uplus \gamma$), $Y\sigma\theta'_1 \equiv Y\theta'_1$ (as $Y \notin dom(\sigma)$). But then no variable in $\mathcal{B}$ can appear in $Y\theta'_1 \equiv Y\theta$ as $(dom(\theta) \cup vran(\theta)) \cap \mathcal{B} = \emptyset$.

  – Let $Y \in dom(\gamma) \setminus dom(\sigma)$ be. Then $Y\gamma \equiv Y(\theta \uplus \gamma) \equiv Y\sigma\theta'_1 \equiv Y\theta'_1$, reasoning like in the previous case. As $dom(\gamma) \subseteq FV(f(\overline{p}) \to s)$ it can happen:

  · $Y \notin FV(f(\overline{p}))$: Then no variable in $\mathcal{B}$ can appear in $Y\gamma$ because $vran(\gamma|_{vExtra(R)}) \cap \mathcal{B} = \emptyset$ by the hypothesis.

- $Y \in FV(f(\overline{p}))$: Let $Z \in \mathcal{B}$ appearing in $Y\gamma$, then Z appears in $f(\overline{t})$, so it must happen $Y \in dom(\sigma)$ because otherwise σ could not be a unifier of $f(\overline{t})$ and $f(\overline{p})$. But this is a contradiction so this case is impossible.

(LetIn) In this case $e\theta \equiv h(e_1\theta, \dots, e\theta, \dots, e_n\theta)$ and $e \equiv h(e_1, \dots, e, \dots, e_n)$. Then the let-rewriting step is

$$e\theta \equiv h(e_1\theta, \dots, e\theta, \dots, e_n\theta) \to^l let\ X = e\theta\ in\ h(e_1\theta, \dots, X, \dots, e_n\theta) \equiv e'$$

with $h \in \Sigma$, $e\theta \equiv f(\overline{e'})$ —$f \in FS$— or $e\theta \equiv let\ Y = e'_1\ in\ e'_2$, and X is a fresh variable. Notice that $e\theta$ is a let-rooted expression or a $f(\overline{e'})$ iff e is a let-rooted expression or a function application, as $\theta \in CTerm$. Then we can apply a let-narrowing step:

$$e \equiv h(e_1, \dots, e, \dots, e_n) \leadsto^l_\sigma let\ X = e\ in\ h(e_1, \dots, X, \dots, e_n) \equiv e''$$

with $\sigma \equiv \epsilon$ and $\theta' \equiv \theta$. Then the conditions in Lemma 11 hold:

i) $e''\theta' \equiv (let\ X = e\ in\ h(e_1, \dots, X, \dots, e_n))\theta \equiv$
$let\ X = e\theta\ in\ h(e_1\theta, \dots, X\theta, \dots, e_n\theta) \equiv$
$let\ X = e\theta\ in\ h(e_1\theta, \dots, X, \dots, e_n\theta) \equiv e'$, since X is fresh an it cannot appear in $dom(\theta')$.
ii) $\sigma\theta' \equiv \epsilon\theta \equiv \theta = \theta[\mathcal{W}]$.
iii) $(dom(\theta') \cup vran(\theta')) \cap \mathcal{B} = (dom(\theta) \cup vran(\theta)) \cap \mathcal{B} = \emptyset$ by hypothesis.

(Bind) In this case $e\theta \equiv let\ X = t\theta\ in\ e_2\theta$ and $e \equiv let\ X = t\ in\ e_2$. Then the let-rewriting step is $let\ X = t\theta\ in\ e_2\theta \to^l e_2\theta[X/t\theta]$ with $t\theta \in CTerm$. As $\theta \in CTerm$, if $t\theta \in CTerm$ then $t \in CTerm$, so we can apply a let-narrowing step:

$$e \equiv let\ X = t\ in\ e_2 \leadsto^l_\sigma e_2[X/t] \equiv e''$$

with $\sigma \equiv \epsilon$ and $\theta' \equiv \theta$. Then the conditions in Lemma 11 hold:

i) $e''\theta' \equiv e_2[X/t]\theta$. By the variable convention we can assume that $X \notin dom(\theta) \cup vran(\theta)$, so by Lemma 1 $e_2[X/t]\theta \equiv e_2\theta[X/t\theta] \equiv e'$.
ii) and iii) As before.

(Elim) We have $e\theta \equiv let\ X = e_1\theta\ in\ e_2\theta$, so $e \equiv let\ X = e_1\ in\ e_2$. Then the let-rewriting step is $e\theta \equiv let\ X = e_1\theta\ in\ e_2\theta \to^l e_2\theta$ with $X \notin FV(e_2\theta)$. By the variable convention $(dom(\theta) \cup vran(\theta)) \cap BV(e) = \emptyset$, so as $X \in BV(e)$ then $X \notin dom(\theta) \cup vran(\theta)$. Then $X \notin FV(e_2\theta)$ implies $X \notin FV(e_2)$ and we can apply a let-narrowing step:

$$e \equiv let\ X = e_1\ in\ e_2 \leadsto^l_\sigma e_2 \equiv e''$$

with $\sigma \equiv \epsilon$ and $\theta' \equiv \theta$. Then the conditions in Lemma 11 hold trivially.

(Flat) In this case $e\theta \equiv let\ X = (let\ Y = e_1\theta\ in\ e_2\theta)\ in\ e_3\theta$ and $e \equiv let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3$. The let-rewriting step is $e\theta \equiv let\ X = (let\ Y = e_1\theta\ in\ e_2\theta)\ in\ e_3\theta \to^l let\ Y = e_1\theta\ in\ let\ X = e_2\theta\ in\ e_3\theta \equiv e'$ with $Y \notin FV(e_3\theta)$.

By a similar reasoning as in the (Elim) case we conclude that $Y \notin dom(\theta) \cup vran(\theta)$, so $Y \notin FV(e_3)$. Then we can apply a let-narrowing step:

$$e \equiv let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \leadsto^l{}_\sigma\ let\ Y = e_1\ in\ let\ X =\ e_2\ in\ e_3 \equiv e''$$

with $\sigma \equiv \epsilon$ and $\theta' \equiv \theta$. Then the conditions in Lemma 11 hold trivially.

(Contx) Then we have $e \equiv \mathcal{C}[s]$. By the variable convention $(dom(\theta) \cup vran(\theta)) \cap BV(e) = \emptyset$, so by lemma 25 $e\theta \equiv (\mathcal{C}[s])\theta \equiv \mathcal{C}\theta[s\theta]$, and the step was

$$e\theta \equiv \mathcal{C}\theta[s\theta] \to^l \mathcal{C}\theta[s'] \equiv e', \text{ because } s\theta \to^l s'$$

Then we know that the lemma holds for $s\theta \to^l s'$, by the proof of the other cases, so taking $\mathcal{W}' = \mathcal{W} \cup FV(s)$ and $\mathcal{B}' = \mathcal{B}$ (as $BV(s) \subseteq BV(\mathcal{C}[s])$) we can do $s \leadsto^l{}_{\sigma_2} s''$ for some θ'_2 under the conditions stipulated. Now we can put this step into (Contx) to do:

$$e \equiv \mathcal{C}[s] \leadsto^l{}_{\sigma_2} \mathcal{C}\sigma_2[s''] \equiv e'' \text{ taking } \sigma = \sigma_2 \text{ and } \theta' = \theta'_2$$

because if $s \leadsto^l{}_{\sigma_2} s''$ was a (Narr) step which lifts a (Fapp) step that uses the fresh variant $(f(\overline{p}) \to r) \in \mathcal{P}$ and adjusts with $\gamma \in CSubst$, then:

- $dom(\sigma_2) \cap BV(\mathcal{C}) = \emptyset$: As $\sigma_2 = mgu(s, f(\overline{p}))$ then $dom(\sigma_2) \subseteq FV(s) \cup FV(f(\overline{p}))$. As $\sigma_2 \lesssim \theta \uplus \gamma$ and it is an mgu then $dom(\sigma_2) \subseteq dom(\theta \uplus \gamma)$. If $X \in FV(s) \cap dom(\sigma_2)$ then $X \notin dom(\gamma) \subseteq FV(f(\overline{p}) \to r)$, so it must happen $X \in dom(\theta)$; but then $X \notin BV(\mathcal{C})$ because $dom(\theta) \cap BV(\mathcal{C}) = \emptyset$ by the variable convention.
  Otherwise it could happen $X \in FV(f(\overline{p})) \cap dom(\sigma_2)$, then X appears in the fresh variant and so it cannot appear in $\mathcal{C}$.
- $vran(\sigma_2|_{\backslash var(\overline{p})}) \cap BV(\mathcal{C}) = \emptyset$: As $dom(\sigma_2) \subseteq FV(s) \cup FV(f(\overline{p}))$ then we have $vran(\sigma_2|_{\backslash var(\overline{p})}) = vran(\sigma_2|_{FV(s)})$. But as $\sigma_2 = mgu(s, f(\overline{p}))$ then $vran(\sigma|_{FV(s)}) \subseteq FV(f(\overline{p}))$, which are part of the fresh variant, so every variable in $vran(\sigma_2|_{\backslash var(\overline{p})})$ is fresh and so cannot appear in $\mathcal{C}$.

Then the conditions in Lemma 11 hold:

ii) $\sigma\theta' = \theta[\mathcal{W}]$: Because $\mathcal{W} \subseteq \mathcal{W}'$, and $\sigma_2\theta'_2 = \theta[\mathcal{W}']$, by the proof of the other cases.

i) $e''\theta' \equiv e'$: As $BV(\mathcal{C}\sigma_2) = BV(\mathcal{C})$, by the variable convention, $BV(\mathcal{C}) \subseteq BV(e) \subseteq BV(\mathcal{B})$, by the hypothesis, and $(dom(\theta'_2) \cup vran(\theta'_2)) \cap \mathcal{B} = \emptyset$, by the proof of the other cases, then $(dom(\theta'_2) \cup vran(\theta'_2)) \cap BV(\mathcal{C}\sigma_2) = \emptyset$. But then:

$$e''\theta' \equiv (\mathcal{C}\sigma_2[s''])\theta'_2 \equiv \underbrace{\mathcal{C}\sigma_2\theta'_2}_{\mathcal{C}\theta}[\underbrace{s''\theta'_2}_{s'}] \equiv e'$$

Because we have $s''\theta'_2 \equiv s'$, by the proof of the other cases, and because $FV(\mathcal{C}) \subseteq FV(e) \subseteq \mathcal{W}$ and $\sigma_2\theta'_2 = \theta[\mathcal{W}]$, as we saw in the previous case (remember $\sigma = \sigma_2$ and $\theta' = \theta'_2$).

iii) $(dom(\theta') \cup vran(\theta')) \cap \mathcal{B} = \emptyset$: Because $\theta' = \theta'_2$ and the proof of the other cases.

The proof for any number of steps proceeds by induction over the number n of steps of the derivation $e\theta \to^{l\,n} e'$. The base case where $n = 0$ is straightforward, as then we have $e\theta \to^{l\,0} e\theta \equiv e'$ so we can do $e \rightsquigarrow^{l\,0}_{\epsilon} e \equiv e''$, so $\sigma = \epsilon$ and taking $\theta' = \theta$ the lemma holds. In the inductive step we have $e\theta \to^{l} e_1 \to^{l^*} e'$, and we will try to build the following diagram:

$$\begin{tikzcd}
e \arrow[r, rightsquigarrow, dotted, "l", "\sigma_1"'] \arrow[d, mapsto, "\theta"'] & e''_1 \arrow[r, rightsquigarrow, dotted, "l^*", "\sigma_2"'] \arrow[d, mapsto, dashed, "\theta_1"'] & e''_2 \equiv e'' \arrow[d, mapsto, dashed, "\theta'_2 = \theta'"] \\
e\theta \arrow[r, "l"'] & e_1 \arrow[r, "*", "l"'] & e'
\end{tikzcd}$$

By the previous proof for one step we have $e \rightsquigarrow^{l}_{\sigma_1} e''_1$ and $\theta'_1 \in CSubst$ under the conditions stipulated. In order to this with the IH we define the sets $\mathcal{B}_1 = \mathcal{B} \cup BV(e_1)$ and $\mathcal{W}_1 = (\mathcal{W} \setminus dom(\sigma_1)) \cup vran(\sigma_1) \cup vE$, where vE is the set of extra variables in the fresh variant $f(\overline{p}) \to s$ used in $e \rightsquigarrow^{l}_{\sigma_1} e''_1$, if it was a (Narr) step; or empty otherwise. We also define $\theta_1 = \theta'_1|_{\mathcal{W}_1}$. Then:

- $FV(e''_1) \cup dom(\theta_1) \subseteq \mathcal{W}_1$: We have $dom(\theta_1) \subseteq \mathcal{W}_1$ by definition of θ_1. On the other hand we have $FV(e''_1) \subseteq \mathcal{W}_1$ because given $X \in FV(e''_1)$ we have two possibilities:

  a) $X \in FV(e))$: then $X \notin dom(\sigma_1)$ since otherwise it disappears in the step $e \rightsquigarrow^{l}_{\sigma_1} e''$. As $dom(\theta) \cup FV(e) \subseteq \mathcal{W}$ then $X \in \mathcal{W} \setminus dom(\sigma_1)$, so $X \in \mathcal{W}_1$.
  b) $X \notin FV(e))$: then there are two possibilities:

     i) X has been inserted by σ_1, so $X \in vran(\sigma_1)$ and $X \in \mathcal{W}_1$.
     ii) X has been inserted as an extra variable in a (Narr) step. Since the narrowing substitution is a mgu then σ_1 cannot affect X, so $X \in \mathcal{W}_1$ because $X \in vE$.

- $e''_1\theta_1 \equiv e_1$: Because as we have seen, $FV(e''_1) \subseteq \mathcal{W}_1$, and so $e''_1\theta_1 \equiv e''_1\theta'_1|_{\mathcal{W}_1} \equiv e''_1\theta'_1 \equiv e_1$, by the proof for one step.
- $BV(e''_1) \subseteq \mathcal{B}_1$: As $\theta'_1 \in CSubst$, $e''_1\theta'_1 \equiv e_1$ and no $CSubst$ can introduce any binding then $BV(e_1) = BV(e''_1)$. But $\mathcal{B}_1 = \mathcal{B} \cup BV(e_1)$, so $BV(e''_1) = BV(e_1) \subseteq \mathcal{B}_1$.
- $(dom(\theta_1) \cup vran(\theta_1)) \cap \mathcal{B}_1 = \emptyset$: As $\theta'_1 \in CSubst$, $e''_1\theta'_1 \equiv e_1$ and no $CSubst$ can introduce any binding then $BV(e_1) = BV(e''_1)$. Then it can happen:

  a) $BV(e''_1) \subseteq BV(e)$: Then $\mathcal{B} = \mathcal{B}_1$, as $BV(e_1) = BV(e''_1) \subseteq BV(e) \subseteq \mathcal{B}$ by hypothesis. Then, as $(dom(\theta'_1) \cup vran(\theta'_1)) \cap \mathcal{B} = \emptyset$ by the proof for one step, then $(dom(\theta'_1) \cup vran(\theta'_1)) \cap \mathcal{B}_1 = \emptyset$, and so $(dom(\theta_1) \cup vran(\theta_1)) \cap \mathcal{B}_1 = \emptyset$, because $\theta_1 = \theta'_1|_{\mathcal{W}_1}$ and so its domain and variable range is smaller than the domain of θ'_1.
  b) $BV(e''_1) \supset BV(e)$: Then $e \rightsquigarrow^{l}_{\sigma_1} e''_1$ must have been a (LetIn) step and so $\sigma = \epsilon$ and $\theta'_1 = \theta$. As the new bounded variable Z is fresh wrt. θ then it is also fresh for $\theta'_1 = \theta$, and so $\mathcal{B}_1 = \mathcal{B} \cup \{Z\}$ has no intersection with $dom(\theta'_1) \cup vran(\theta'_1)$ nor with $dom(\theta_1) \cup vran(\theta_1)$, which is smaller.

- $\sigma_1\theta_1 = \theta[\mathcal{W}]$: It is enough to see that $\sigma_1\theta_1 = \sigma_1\theta'_1[\mathcal{W}]$, because we have $\sigma_1\theta'_1 = \theta[\mathcal{W}]$ by the proof for one step, and this is true because given $X \in \mathcal{W}$:

  a) If $X \in dom(\sigma_1)$ then $FV(X\sigma_1) \subseteq vran(\sigma_1) \subseteq \mathcal{W}_1$, so as $\theta_1 = \theta'_1|_{\mathcal{W}_1}$ then $X\sigma_1\theta_1 \equiv X\sigma_1\theta'_1|_{\mathcal{W}_1} \equiv X\sigma_1\theta'_1$.
  b) If $X \in \mathcal{W} \setminus dom(\sigma_1)$ then $X \in \mathcal{W}_1$ by definition, and so $X\sigma_1\theta_1 \equiv X\theta_1$ (as $X \notin dom(\sigma_1)$), $X\theta_1 \equiv X\theta'_1|_{\mathcal{W}_1} \equiv X\theta'_1$ (as $X \in \mathcal{W}_1$), and $X\theta'_1 \equiv X\sigma\theta'_1$ (as $X \notin dom(\sigma_1)$).

So we have $e''_1\theta_1 \equiv e_1$ and $e_1 \to^{l^*} e'$, but then we can apply the induction hypothesis to $e''_1\theta_1 \to^{l^*} e'$ using $\mathcal{W}_1$ and $\mathcal{B}_1$, which fulfill the hypothesis of the lemma, as we have seen. Then we get $e''_1 \leadsto^{l*}_{\sigma_2} e''_2$ and $\theta'_2 \in CSubst$ under the conditions stipulated. But then we have:

$$e \leadsto^{l}_{\sigma_1} e''_1 \leadsto^{l*}_{\sigma_2} e''_2 \text{ taking } e'' \equiv e''_2,\ \sigma = \sigma_1\sigma_2 \text{ and } \theta' = \theta'_2$$

for which we can prove the conditions in Lemma 11:

i) $e''\theta' \equiv e'$: As $e''\theta' \equiv e''_2\theta'_2 \equiv e'$ by IH.

ii) $\sigma\theta' = \theta[\mathcal{W}]$: That is, $\sigma_1\sigma_2\theta'_2 = \theta[\mathcal{W}]$. As we have $\sigma_1\theta_1 = \theta[\mathcal{W}]$, as we saw before, all that is left is proving $\sigma_1\sigma_2\theta'_2 = \sigma_1\theta_1[\mathcal{W}]$, which happens because given $X \in \mathcal{W}$:

  a) If $X \in dom(\sigma_1)$ then $FV(X\sigma_1) \subseteq vran(\sigma_1) \subseteq \mathcal{W}_1$, so as $\sigma_2\theta'_2 = \theta_1[\mathcal{W}_1]$ by IH, then $(X\sigma_1)\sigma_2\theta'_2 \equiv (X\sigma_1)\theta_1$.
  b) If $X \in \mathcal{W} \setminus dom(\sigma_1)$ then $X \in \mathcal{W}_1$ by definition, and so, as $\sigma_2\theta'_2 = \theta_1[\mathcal{W}_1]$ by IH, then $X\sigma_1\sigma_2\theta'_2 \equiv X\sigma_2\theta'_2$ (as $X \notin dom(\sigma_1)$), $X\sigma_2\theta'_2 \equiv X\theta_1$ (as $X \in \mathcal{W}_1$), $X\theta_1 \equiv X\sigma_1\theta_1$ (as $X \notin dom(\sigma_1)$).

iii) $(dom(\theta') \cup vran(\theta')) \cap \mathcal{B} = \emptyset$: That is $(dom(\theta'_2) \cup vran(\theta'_2)) \cap \mathcal{B} = \emptyset$, which happens as $(dom(\theta'_2) \cup vran(\theta'_2)) \cap \mathcal{B}_1 = \emptyset$ by IH and $\mathcal{B} \subseteq \mathcal{B}_1$.

□

A.10 Proofs for Section 7

The let-binding elimination transformation $\widehat{\ }$ satisfies the following interesting properties, which illustrate that its definition is sound.

Lemma 33
For all $e, e' \in LExp$, $\mathcal{C} \in Cntxt$, $X \in \mathcal{V}$ we have:

i) $|\widehat{e}| \equiv |e|$.
ii) If $e \in Exp$ then $\widehat{e} \equiv e$.
iii) $FV(\widehat{e}) \subseteq FV(e)$
iv) $\widehat{e[X/e']} = \widehat{e}[X/\widehat{e'}]$.

Proof

i–iii) Easily by induction on the structure of e.

iv) A trivial induction on the structure of e, using Lemma 1 for the case when e has the shape $e \equiv let\ X = e_1\ in\ e_2$.

□

Lemma 12 (Copy lemma)
For all $e, e_1, e_2 \in Exp$, $X \in \mathcal{V}$:

i) $e_1 \to e_2$ implies $e[X/e_1] \to^* e[X/e_2]$.
ii) $e_1 \to^* e_2$ implies $e[X/e_1] \to^* e[X/e_2]$.

Proof
To prove *i)* we proceed by induction on the structure of e. Concerning the base cases:

- If $e \equiv X$ then $e[X/e_1] \equiv e_1 \to e_2 \equiv e[X/e_2]$, by hypothesis.
- If $e \equiv Y \in \mathcal{V} \setminus \{X\}$ then $e[X/e_1] \equiv Y \to^0 Y \equiv e[X/e_2]$.
- Otherwise $e \equiv h$ for some $h \in \Sigma$, so $e[X/e_1] \equiv h \to^0 h \equiv e[X/e_2]$

Regarding the inductive step, then $e \equiv h(e'_1, \dots, e'_n)$ and so

$$\begin{array}{ll} e[X/e_1] \equiv h(e'_1[X/e_1], \dots, e'_n[X/e_1]) & \\ \to^* h(e'_1[X/e_2], \dots, e'_n[X/e_2]) & \text{by IH, } n \text{ times} \\ \equiv e[X/e_2] & \end{array}$$

The proof for *ii)* follows the same structure. □

Lemma 13 (One-Step Soundness of let-rewriting wrt. term rewriting)
For all $e, e' \in LExp$ we have that $e \to^l e'$ implies $\widehat{e} \to^* \widehat{e'}$.

Proof
We proceed by a case distinction over the rule of let-rewriting used in the step $e \to^l e'$.

(Fapp) Then we have:

$$e \equiv f(\overline{p})\theta \to^l r\theta \equiv e' \text{ for some } (f(\overline{p}) \to r) \in \mathcal{P}, \theta \in CSubst$$

But then $f(\overline{p})\theta, r\theta \in Exp$, therefore $\widehat{f(\overline{p})\theta} \equiv f(\overline{p})\theta$ and $\widehat{r\theta} \equiv r\theta$, by Lemma 33 *ii)*, and so we can link $\widehat{e} \equiv \widehat{f(\overline{p})\theta} \equiv f(\overline{p})\theta \to r\theta \equiv \widehat{r\theta} \equiv \widehat{e'}$, by a term rewriting step.

(LetIn) Then we have:

$$e \equiv h(e_1, \dots, e_k, \dots, e_n) \to^l let\ X = e_k\ in\ h(e_1, \dots, X, \dots, e_n) \equiv e'$$

where X is a fresh variable (among other conditions). But then

$$\begin{array}{ll} \widehat{e'} \equiv \widehat{h(e_1, \dots, X, \dots, e_n)}[X/\widehat{e_k}] \equiv h(\widehat{e_1}, \dots, X, \dots, \widehat{e_n})[X/\widehat{e_k}] & \\ \equiv h(\widehat{e_1}, \dots, \widehat{e_k}, \dots, \widehat{e_n}) & \text{as } X \text{ is fresh} \\ \equiv \widehat{h(e_1, \dots, e_k, \dots, e_n)} \equiv \widehat{e} & \end{array}$$

Therefore $\widehat{e} \to^0 \widehat{e} \equiv \widehat{e'}$.

(Bind) Then we have:

$$e \equiv let\ X = t\ in\ e_1 \to^l e_1[X/t] \equiv e' \text{ with } t \in CTerm$$

But then $\widehat{e} \equiv \widehat{e_1}[X/\widehat{t}] \equiv \widehat{e_1[X/t]} \equiv \widehat{e'}$, by Lemma 33 *iv)*, hence $\widehat{e} \to^0 \widehat{e} \equiv \widehat{e'}$.

(Elim) Then we have:

$$e \equiv let\ X = e_1\ in\ e_2 \to^l e_2 \equiv e' \text{ with } X \notin FV(e_2)$$

But then

$$\begin{array}{ll} \widehat{e} \equiv \widehat{e_2}[X/\widehat{e_1}] & \\ \equiv e_2\widehat{[X/e_1]} & \text{by Lemma 33 } iv) \\ \equiv \widehat{e_2} \equiv \widehat{e'} & \text{as } X \notin FV(e_2) \end{array}$$

Therefore $\widehat{e} \to^0 \widehat{e} \equiv \widehat{e'}$.

(Flat) Then we have:

$$e \equiv let\ X = (let\ Y = e_1\ in\ e_2)\ in\ e_3 \to^l let\ Y = e_1\ in\ (let\ X = e_2\ in\ e_3) \equiv e'$$

where $Y \notin FV(e_3)$. But then

$$\begin{array}{ll} \widehat{e} \equiv \widehat{e_3}[X/\widehat{let\ Y = e_1\ in\ e_2}] \equiv \widehat{e_3}[X/(\widehat{e_2}[Y/\widehat{e_1}])] & \\ \equiv \widehat{e_3}[X/\widehat{e_2}][Y/\widehat{e_1}] & Y \notin FV(\widehat{e_3}) \text{ by Lemma 33 } iii) \\ \equiv (\widehat{let\ X = e_2\ in\ e_3})[Y/\widehat{e_1}] \equiv \widehat{e'} & \end{array}$$

Therefore $\widehat{e} \to^0 \widehat{e} \equiv \widehat{e'}$.

(Contx) Then we have:

$$e \equiv \mathcal{C}[e_1] \to^l \mathcal{C}[e_2] \equiv e'$$

with $e_1 \to^l e_2$ by some of the previous rules, therefore $\widehat{e_1} \to^* \widehat{e_2}$ by the proof of the previous cases. We will prove that $\widehat{e_1} \to^* \widehat{e_2}$ implies $\widehat{\mathcal{C}[e_1]} \to^* \widehat{\mathcal{C}[e_2]}$, thus getting $\widehat{e} \to^* \widehat{e'}$ as a trivial consequence.

We proceed by induction on the structure of $\mathcal{C}$. Regarding the base case then $\mathcal{C} \equiv []$ and so $\widehat{\mathcal{C}[e_1]} \equiv \widehat{e_1} \to^* \widehat{e_2} \equiv \widehat{\mathcal{C}[e_2]}$ by hypothesis. For the inductive step:

- If $\mathcal{C} \equiv let\ X = \mathcal{C}'\ in\ a$ then by IH we get $\widehat{\mathcal{C}'[e_1]} \to^* \widehat{\mathcal{C}'[e_2]}$, and so

  $$\begin{array}{l} \widehat{\mathcal{C}[e_1]} \equiv \widehat{a}[X/\widehat{\mathcal{C}'[e_1]}] \\ \to^* \widehat{a}[X/\widehat{\mathcal{C}'[e_2]}] \text{ by IH and Lemma 12} \\ \equiv \widehat{\mathcal{C}[e_2]} \end{array}$$

  Notice that it is precisely because of this case that we cannot say that $e \to^l e'$ implies $\widehat{e} \to^* \widehat{e'}$ in zero or one steps, because the copies of $\widehat{\mathcal{C}'[e_1]}$ made by the substitution $[X/\widehat{\mathcal{C}'[e_1]}]$ may force the zero or one steps derivation from $\widehat{\mathcal{C}'[e_1]}$ to be repeated several times in derivation $\widehat{a}[X/\widehat{\mathcal{C}'[e_1]}] \to^* \widehat{a}[X/\widehat{\mathcal{C}'[e_2]}]$. This is typical situation when mimicking term graph rewriting derivations by term rewriting.
- If $\mathcal{C} \equiv let\ X = a\ in\ \mathcal{C}'$ then $\widehat{\mathcal{C}[e_1]} \equiv \widehat{\mathcal{C}'[e_1]}[X/\widehat{a}] \to^* \widehat{\mathcal{C}'[e_2]}[X/\widehat{a}] \equiv \widehat{\mathcal{C}[e_2]}$, by IH combined with closedness under substitutions of term rewriting.

- Otherwise $\mathcal{C} \equiv h(a_1, \ldots, \mathcal{C}', \ldots, a_n)$ and then $\widehat{\mathcal{C}[e_1]} \equiv h(\widehat{a_1}, \ldots, \widehat{\mathcal{C}'[e_1]}, \ldots, \widehat{a_n})$ $\to^* h(\widehat{a_1}, \ldots, \widehat{\mathcal{C}'[e_2]}, \ldots, \widehat{a_n}) \equiv \widehat{\mathcal{C}[e_2]}$ by IH.

□

Proposition 10
For all $\sigma \in Subst_\perp$, $\theta \in [\![\sigma]\!]$, we have that $\theta \trianglelefteq \sigma$.

Proof
Given some $X \in \mathcal{V}$, we have two possibilities. If $X \in dom(\theta)$ then taking any $t \in CTerm_\perp$ such that $\mathcal{P} \vdash_{CRWL} \theta(X) \to t$, by Lemma 5 we have $t \sqsubseteq \theta(X)$, because $\theta \in [\![\sigma]\!] \subseteq CSubst_\perp$. But $\theta \in [\![\sigma]\!]$ implies $\mathcal{P} \vdash_{CRWL} \sigma(X) \to \theta(X)$, therefore $\mathcal{P} \vdash_{CRWL} \sigma(X) \to t$ by the polarity from Proposition 3, which holds for CRWL too. Hence $[\![\theta(X)]\!] \subseteq [\![\sigma(X)]\!]$.

On the other hand, if $X \not\in dom(\theta)$ then for any $t \in CTerm_\perp$ such that $\mathcal{P} \vdash_{CRWL}$ $\theta(X) \equiv X \to t$ we have that $t \equiv \perp$ or $t \equiv X$. If $t \equiv \perp$ then $\mathcal{P} \vdash_{CRWL} \sigma(X) \to t$ by rule (B). Otherwise $\theta \in [\![\sigma]\!]$ implies $\mathcal{P} \vdash_{CRWL} \sigma(X) \to \theta(X) \equiv X \equiv t$. Hence $[\![\theta(X)]\!] \subseteq [\![\sigma(X)]\!]$. □

Proposition 11
For all $\sigma \in DSusbt_\perp$, $[\![\sigma]\!]$ is a directed set.

Proof
For any preorder $\leq$, any directed set D wrt. it and any elements $e_1, e_2 \in D$ by $e_1 \sqcup_D e_2$ we denote the element $e_3 \in D$ such that $e_1 \leq e_3$ and $e_2 \leq e_3$ that must exist because D is directed.

Now, given any $\sigma \in DSubst_\perp$ we have that $\forall X \in \mathcal{V}, [\![\sigma(X)]\!]$ is a directed set, because if $X \in dom(\sigma)$ then we can apply the definition of $DSubst_\perp$ and otherwise $[\![X]\!] = \{X, \perp\}$, which is directed. Now given $\theta_1, \theta_2 \in [\![\sigma]\!]$ we can define $\theta_3 \in CSubst_\perp$ as $\theta_3(X) = \theta_1(X) \sqcup_{\sigma(X)} \theta_2(X)$, which fulfills:

1. $\theta_i \sqsubseteq \theta_3$ for $i \in \{1, 2\}$, because for any $X \in \mathcal{V}$ we have that $[\![\sigma(X)]\!]$ is directed (as we saw above) and $\theta_i(X) \in [\![\sigma(X)]\!]$ (because $\theta_1, \theta_2 \in [\![\sigma]\!]$), therefore $\theta_i(X) \sqsubseteq \theta_1(X) \sqcup_{\sigma(X)} \theta_2(X) = \theta_3(X)$ by definition.
2. $\theta_3 \in [\![\sigma]\!]$, because $\forall X \in \mathcal{V}, \theta_3(X) = \theta_1(X) \sqcup_{\sigma(X)} \theta_2(X) \in [\![\sigma(X)]\!]$ by definition.

□

We will use the following lemma about non-triviality of substitution denotations as an auxiliary result for proving Lemma 15.

Lemma 34
For all $\sigma \in Subst_\perp$ we have that $[\![\sigma]\!] \neq \emptyset$ and given $\overline{X} = dom(\sigma)$ then $[\overline{X/\perp}] \in [\![\sigma]\!]$.

Proof
It is enough to prove that if $\overline{X} = dom(\sigma)$ then $[\overline{X/\perp}] \in [\![\sigma]\!]$. First of all $[\overline{X/\perp}] \in CSubst_\perp$ by definition. Now consider some $Y \in \mathcal{V}$.

i) If $Y \in \overline{X}$ then $\sigma(Y) \to \perp \equiv Y[\overline{X/\perp}]$, by rule (B).
ii) Otherwise $Y \not\in \overline{X} = dom(\sigma)$, hence $\sigma(Y) \equiv Y \to Y \equiv Y[\overline{X/\perp}]$, by rule (RR).

□

Lemma 15
For all $\sigma \in DSusbt_\perp$, $e \in Exp_\perp, t \in CTerm_\perp$,

$$\text{if } e\sigma \rightarrowtriangle t \text{ then } \exists\theta \in [\![\sigma]\!] \text{ such that } e\theta \rightarrowtriangle t$$

Proof
We proceed by a case distinction over e:

- If $e \equiv X \in dom(\sigma)$: Then $e\sigma \equiv \sigma(X) \rightarrowtriangle t$, so we can define:

$$\theta(Y) = \begin{cases} t & \text{if } Y \equiv X \\ \perp & \text{if } Y \in dom(\sigma) \setminus \{X\} \\ Y & \text{otherwise} \end{cases}$$

Then $\theta \in [\![\sigma]\!]$ because obviously $\theta \in CSusbt_\perp$, and given $Z \in \mathcal{V}$.

a) If $Z \equiv X$ then $\sigma(Z) \equiv \sigma(X) \rightarrowtriangle t \equiv \theta(Z)$ by hypothesis.
b) If $Z \in (dom(\sigma) \setminus \{X\})$ then $\sigma(Z) \rightarrowtriangle\perp\equiv \theta(Z)$ by rule (B).
c) Otherwise $Z \notin dom(\sigma)$ and then $\sigma(Z) \equiv Z \rightarrowtriangle Z \equiv \theta(Z)$ by rule (RR).

But then $e\theta \equiv \theta(X) \equiv t \rightarrowtriangle t$ by Lemma 5—which also holds for CRWL, because CRWL and CRWL$_{let}$ coincide for c-terms— , as $t \in CTerm_\perp$.

- If $e \equiv X \notin dom(\sigma)$: Then given $\overline{Y} = dom(\sigma)$ we have $[\overline{Y/\perp}] \in [\![\sigma]\!]$ by Lemma 34, so we can take $\theta = \{[\overline{Y/\perp}]\}$ for which $[\![e\sigma]\!] = [\![X\sigma]\!] = [\![X]\!] = [\![X[\overline{Y/\perp}]]\!] = [\![X\theta]\!]$.
- If $e \notin \mathcal{V}$ then we proceed by induction over the structure of $e\sigma \rightarrowtriangle t$:

Base cases

(B) Then $t \equiv\perp$, so given $\overline{Y} = dom(\sigma)$ we can take $\theta = \{[\overline{Y/\perp}]\}$ for which $e\theta \rightarrowtriangle\perp$ by rule (B).
(RR) Then $e \in \mathcal{V}$ and we are in the previous case.
(DC) Similar to the case for $e \equiv X \notin dom(\sigma)$.

Inductive steps

(DC) Then $e \equiv c(e_1, \ldots, e_n)$, as $e \notin \mathcal{V}$, and we have:

$$\frac{e_1\sigma \rightarrowtriangle t_1 \ \ldots\ e_n\sigma \rightarrowtriangle t_n}{e\sigma \equiv c(e_1\sigma, \ldots, e_n\sigma) \rightarrowtriangle c(t_1, \ldots, t_n) \equiv t}\ DC$$

Then by IH or the proof of the other cases we have that $\forall i \in \{1, \ldots, n\}.\ \exists\theta_i \in [\![\sigma]\!]$ such that $e_i\theta_i \rightarrowtriangle t_i$. But as $\sigma \in DSusbt_\perp$ then $[\![\sigma]\!]$ is directed by Lemma 11, therefore there must exist some $\theta \in [\![\sigma]\!]$ such that $\forall i \in \{1, \ldots, n\}.\theta_i \sqsubseteq \theta$, and so by Proposition 5 —which also holds for CRWL, by Theorem 4— we have $\forall i \in \{1, \ldots, n\}.e_i\theta \rightarrowtriangle t_i$, so we can build the following proof:

$$\frac{e_1\theta \rightarrowtriangle t_1 \ \ldots e_n\theta \rightarrowtriangle t_n}{e\theta \equiv c(e_1\theta, \ldots, e_n\theta) \rightarrowtriangle c(t_1, \ldots, t_n) \equiv t}\ DC$$

(OR) Very similar to the proof of the previous case. We also have $e \equiv f(e_1, \ldots, e_n)$ (as $e \notin \mathcal{V}$) and given a proof for $e\sigma \equiv f(e_1, \ldots, e_n)\sigma \rightarrowtriangle t$, so we can apply the IH or the proof of the other cases to every $e_i\sigma \rightarrowtriangle p_i\mu$ to get some $\theta_i \in [\![\sigma]\!]$

such that $e_i\theta_i \rightarrowtriangle p_i\mu$. Then we can use Lemma 11 and Proposition 5 to use the obtained θ to compute the same values for the arguments of f, thus using the same substitution $\mu \in CSubst_\perp$ for parameter passing in (OR).

□

Theorem 19

Let $\mathcal{P}$ be a CRWL-deterministic program, and $e, e' \in Exp, t \in CTerm$. Then:

a) $e \to^* e'$ implies $e \to^{l^*} e''$ for some $e'' \in LExp$ with $|e''| \sqsupseteq |e'|$.
b) $e \to^* t$ iff $e \to^{l^*} t$ iff $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$.

Proof

a) Assume $e \to^* e'$. By Lemma 16, $[\![e']\!] \subseteq [\![e]\!]$ and by Lemma 5 we have $|e'| \in [\![e']\!]$, then $|e'| \in [\![e]\!]$. Therefore, by Theorem 12 there exists $e'' \in LExp$ such that $e \to^{l^*} e''$ with $|e''| \sqsupseteq |e'|$.
b) The parts $e \to^{l^*} t$ iff $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$, and $e \to^{l^*} t$ implies $e \to^* t$ have been already proved for arbitrary programs in Theorems 12 and 17 respectively. What remains to be proved is that $e \to^* t$ implies $e \to^{l^*} t$ (or the equivalent $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$). Assume $e \to^* t$. Then $[\![t]\!] \subseteq [\![e]\!]$ by Lemma 16. Now, by Lemma 5 $t \in [\![t]\!]$, and therefore $t \in [\![e]\!]$, which exactly means that $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$.

□